CHAT: AUTOMATED, CONSTRAINED AND HARD-LABEL ATTACKS ON TABULAR MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Evaluating tabular models' adversarial robustness is particularly challenging, as input perturbations must satisfy domain constraints (i.e., semantic relationships between features) to reflect realistic samples. Most existing attacks rely on manually created constraints which are inherently limited. Moreover, as they rely on model gradients or output scores, existing attacks are inadequate for practical settings where only hard labels are available. We address the first challenge by leveraging a large language model (LLM) to automatically mine rich, high-quality constraints. Our evaluation of constraints derived for three datasets from TabularBench shows that LLM-generated constraints almost fully capture manual constraints, cover additional critical constraints not found manually, and score highly on quality assessment by domain experts. We address the second challenge by proposing the first hard-label practical black-box attack on tabular models. By interweaving a misclassification-aware projection method for satisfying constraints and a leading hard-label attack from vision (CGBA), our attack, termed CHAT, achieves higher constrained attack success rates than the state-of-the-art white-box attack (CAA) on three datasets and eight models each, both defended and undefended.

1 INTRODUCTION

Tabular models are commonly used in critical domains such as finance and security (Simonetto et al., 2024b). Hence, it is critical to reliably evaluate their adversarial robustness to evasive inputs—i.e. adversarial examples (Szegedy et al., 2014; Goodfellow et al., 2015). Yet, such evaluations are obstructed by several challenges: *(1)* To ensure adversarial examples pose a realistic risk, they must satisfy domain-specific relational constraints (e.g., a subset of features may need to sum up to one), but these are often defined manually (Simonetto et al., 2024b; 2022; Chernikova & Oprea, 2022) or lack comprehensiveness and precision (Ben-Tov et al., 2024), *(2)* Attempts to satisfy relational constraints may harm other attack objectives (mainly, adversarial example candidates may no longer be misclassified, App. B.6); and *(3)* Evaluation should simulate the realistic setting where only hard labels are returned by black-box models (Brendel et al., 2018; Chen et al., 2020).

Our work tackles these problems by proposing the CHAT attack framework (§4). CHAT leverages large language models (LLMs) to automate the mining of comprehensive, high-quality constraints (§4.1). For comprehensiveness, CHAT iterates over all features and queries the LLM for constraints involving each feature multiple times. Subsequently, CHAT searches for and attempts to repair possibly inaccurate constraints to maintain quality. Our evaluation on three TabularBench datasets (Simonetto et al., 2024b) indeed shows that the LLM-generated constraints capture human-defined, so-called golden constraints as well as other critical ones undiscovered by humans (§5.1).

Off-the-shelf methods used to project adversarial example candidates often fail to preserve misclassification or satisfy constraints. To produce realistic adversarial examples that satisfy constraints, CHAT uses a cyclic projection (constraint repair) method that aims to iteratively satisfy constraints (§4.2). Notably, CHAT's projection incorporates LLM assistance to simplify and reformulate constraints for solvers and misclassification-aware recovery to propose alternative attack candidates when projection hinders misclassification.

Existing attacks for tabular models are primarily tailored for white- or grey-box settings, where models are fully accessible (Simonetto et al., 2024a; Ben-Tov et al., 2024) or expose prediction scores (Simonetto et al., 2022), respectively. We develop a variant of CHAT tailored for hard-label black-box

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

settings by adapting CGBA (Reza et al. (2023)), a hard-label attack from vision. Particularly, we find that CGBA often gets stuck in local optima when tested on tabular models. Thus, we extend it by encouraging exploration through allowing temporary relaxation of the adversarial example's norm and enriching the search directions for evasive inputs (§4.3).

We combine CHAT's projection with established white-box methods for producing evasive inputs (App. B.1; yielding the CHAT_{WB} variant) as well as our adaptation of CGBA to tabular models under hard-label black-box settings (yielding CHAT_{BB}). We evaluate both variants on three datasets with eight models each, both defended and undefended, and find that, in most settings tested, both CHAT_{WB} and CHAT_{BB} attain higher constrained attack success rate—i.e., portion of evasive adversarial examples that also satisfy constraints—compared to the state-of-the-art white-box attack CAA (Simonetto et al., 2024a) when fed with identical constraints (§5.2).

2 Related Work

Tabular attacks. Several attacks have been proposed for generating realistic adversarial examples satisfying domain constraints against tabular models through norm-bounded perturbations (Pierazzi et al., 2020). CaFA (Ben-Tov et al., 2024) mines logical constraints, perturbs inputs to evade classification, and projects them to create evasive samples. CSP (Sheatsley et al., 2021) follows a similar approach but uses a less expressive constraint formalism (Ben-Tov et al., 2024). CAPGD (Simonetto et al., 2024a) satisfies manual constraints by formulating them as objectives during optimization, MOEVA (Simonetto et al., 2022) uses an evolutionary approach to produce attacks, and CAA (Simonetto et al., 2024a) interweaves the two. URET (Eykholt et al., 2023) and FENCE (Chernikova & Oprea, 2022) employ manually defined transformations to satisfy constraints while transforming samples. Bostani et al. (2025) produce adversarial examples that preserve feature dependencies found through correlation analysis.

Table 5 in App. A summarizes the differences between CHAT and prior work across multiple axes. Most attacks are applicable under permissive white-box settings (Ben-Tov et al., 2024; Simonetto et al., 2024a; Sheatsley et al., 2021; Eykholt et al., 2023; Bostani et al., 2025; Chernikova & Oprea, 2022), where models expose weights and gradients. By contrast, MOEVA is also applicable under stricter settings where models return classification scores (Simonetto et al., 2022). Differently, CHAT aims to tackle the least permissive and most practical setting, where models only return hard labels.

Existing attacks often rely on manually defined constraints (Simonetto et al., 2024a; 2022) or transformations that do not violate domain constraints (Eykholt et al., 2023; Chernikova & Oprea, 2022). By contrast, approaches that automate constraint mining may fall short of capturing manual constraints (Ben-Tov et al., 2024; Sheatsley et al., 2021; Bostani et al., 2025), as their formalism may not be sufficiently rich. As we show in §5.1, CHAT's constraints, generated using LLMs, are better at capturing manual constraints and beyond.

In short, to our knowledge, CHAT is the first attack for generating realistic adversarial examples for tabular models through automatically mined constraints that also lends itself for assessing robustness under restricted hard-label black-box settings.

Hard-label attacks in other domains. Multiple decision-based attacks tailored for the hard-label black-box settings have been proposed in other domains, primarily in vision (e.g., Brendel et al. (2018); Chen et al. (2020)). We instantiate CHAT_{BB} using CGBA (Reza et al., 2023) due to its efficiency. CGBA traverses the classification boundary through search along the estimated boundary normal with curvature-aware search along semi-circular paths in two dimensional planes. We find that vanilla CGBA fails to escape local optima encountered in low-dimensional tabular domains and propose solutions to enhance it.

App. A presents additional related work and provides more details.

3 Problem Setup and Threat Model

Let $f : \mathcal{X} \to \mathcal{Y}$ be a tabular model and (x, y) a correctly classified sample (i.e., $f(x) = y$). The adversary seeks an (untargeted) adversarial example x' satisfying $f(x') \neq y$, which is close to the clean sample x in a distance metric $D(\cdot, \cdot)$ (i.e., $D(x, x') \leq \epsilon$). We consider scaled L_2 distance,

where numerical features are min-max normalized to $[0, 1]$, in line with prior work (Simonetto et al., 2024b).

The attacker receives structural constraints $\mathcal{C}_{\text{struct}}$ specifying feature types, ranges, categorical domains, and immutability. A constraint is a Boolean predicate $c(z; x)$, where dependence on the original sample x permits restrictions on allowed modifications. For any constraint set $\mathcal{C}$, define

$$\mathcal{F}(\mathcal{C}; x) = \{z \in \mathcal{X} : c(z; x) = 1 \text{ for every } c \in \mathcal{C}\}.$$

Adversarial examples must satisfy the structural constraints, i.e., $x' \in \mathcal{F}(\mathcal{C}_{\text{struct}}; x)$.

We consider attacks under both white- and black-box settings. In both attack settings, the attacker receives the schema information (i.e., feature descriptions), necessary for using the model, and a held-out set of unlabeled samples. The attacker must infer the feature interdependencies that valid adversarial examples should satisfy from this information. We denote the manually specified golden constraints by $\mathcal{C}_{\text{gold}}$. These constraints are hidden from the attacker during both constraint generation and attack construction and are used only for evaluation.

4 CHAT

CHAT has three components: constraint generation using LLMs, cyclic projection with recovery, and extension to hard-label black-box settings. Using the schema information and held-out clean samples, CHAT first mines relational constraints ($\mathcal{C}_{\text{LLM}}$ denotes the retained set). Once mining is complete, the attack seeks an adversarial example x' satisfying

$$f(x') \neq y, \qquad D(x, x') \leq \epsilon, \qquad x' \in \mathcal{F}(\mathcal{C}_{\text{struct}} \cup \mathcal{C}_{\text{LLM}}; x). \tag{1}$$

For benchmark evaluation, we additionally assess the same misclassification and distance conditions together with feasibility under $\mathcal{C}_{\text{struct}} \cup \mathcal{C}_{\text{gold}}$. We report success under $\mathcal{C}_{\text{gold}}$ and $\mathcal{C}_{\text{LLM}}$ separately.

4.1 Constraint Generation

Example, step 0: Setup and classifier. Consider synthetic rows (O, T, C, r, V): open, total, and closed account counts, the open-account ratio, and the verified-account count. This is an illustrative example. Let

$$x_{\texttt{origin}} = (2, 10, 8, 0.2, 2), \qquad f(z) = \mathbf{1}[2O - V \geq 5].$$

The original label is 0; an adversarial row must have $2O - V \geq 5$. The perturbation budget is $\epsilon = 0.5$ under range-normalized distance from $x_{\texttt{origin}}$. App. C.4 gives the schema, metric, and full calculations.

We perform LLM-assisted constraint generation iteratively across features using o3 (App. B.2). For feature i, we construct a textual description s_i containing its name, semantic meaning, type (real-valued, integer-valued, or categorical), range, units (USD, kg/m^2, etc.), the meaning of categorical values, and three clean samples. These descriptions provide semantic information required to infer relationships between features. We then loop over each feature k times, and for every feature i, we query the LLM for a constraint using a prompt containing s_i and the descriptions of all other features. The prompt asks for relational constraints involving feature i, returned as a Python boolean validator alongside a short description. To avoid contamination, we use a model whose training data cutoff predates the evaluation benchmark (App. C). For n features, this process yields an initial set of $n \times k$ candidate constraints, with potentially redundant or incorrect ones.

Example, step 1: Generate. For the open-account feature O, an LLM proposes $O < T$: the number of open accounts is strictly smaller than the total.

To facilitate correctness, we take several measures. First, we execute every candidate constraint c on a held-out clean set $\mathcal{D}_{\text{hold}}$ and require

$$\frac{1}{|\mathcal{D}_{\text{hold}}|} \sum_{x \in \mathcal{D}_{\text{hold}}} \mathbf{1}[c(x) = 1] \geq 1 - \tau,$$

where $\tau \approx 0$ tolerates rare data errors and extreme cases that could otherwise discard useful constraints. For a failing constraint, we re-query the LLM with the description, code, errors, and both passing and failing samples, and ask for a minimal repair, discarding constraints after failed repairs or LLM refusal. We attempt only one repair to prevent overfitting the observed samples or introducing

loose tolerances. After k iterations, the LLM is queried to simplify constraints or propose additional ones. Each change or addition also undergoes validation and repair.

Example, step 2: Validate and refine. A clean row with $O = T = 5$ rejects $O < T$; one repair gives $O \leq T$. Refinement adds $c_R : r = O/T$ for $T > 0$, with $O = r = 0$ at $T = 0$. Other feature queries yield $c_P : T = O + C$ and $c_V : O \leq V$.

As we generate each constraint independently from previous ones, redundant ones might emerge.[1] To address this, we attempt to remove redundant constraints. Using dense retrieval, we select, for each constraint c, the most similar constraints by its description. For each retrieved constraint c_i, an SMT (satisfiability modulo theories) solver (de Moura & Bjørner, 2008) checks whether c implies c_i. We remove c_i as redundant only if it is implied by a retained constraint.

Finally, we query the LLM to judge if the constraint is semantically meaningful and correct; if the response is negative, we remove it.

Example, step 3: Remove redundancy. Since $C \geq 0$, the retained $c_P : T = O + C$ implies $O \leq T$. The weaker bound is removed. After the semantic checks, three distinct relationships remain: the ratio c_R, partition c_P, and verification rule c_V.

App. C.1 presents the complete constraint-generation algorithm.

4.2 Constraint Projection

In both white- and black-box settings, CHAT first generates an attack candidate subject to structural constraints and the perturbation budget, then projects onto $\mathcal{C}_{\text{LLM}}$. A single perturbation–projection cycle suffices if successful. If the projection procedure cannot recover a valid adversarial example, a restart step, described below, freezes a selected feature, resumes the base attack, and reattempts projection.

When the base attack first generates an attack candidate, it may violate $\mathcal{C}_{\text{LLM}}$, while repairing violations may restore correct prediction. To address this, CHAT combines cyclic feasibility repair with misclassification-aware recovery and LLM assistance to seek a point satisfying Eq. 1.

Approximate cyclic repair (projection). Constraints often involve a small number of features. Cyclic projection exploits this sparsity by solving small local problems and incorporating interacting constraints as needed. Starting from an attack candidate, we select a violated relation and optimize only its mutable features, keeping all other features (as defined in $\mathcal{C}_{\text{struct}}$) fixed. Each local repair balances between proximity to the current candidate x and proximity to the original sample $x_{\texttt{origin}}$:

$$\begin{aligned} \min_{x'} \quad & \tfrac{1}{2}\|W(x' - x)\|_2^2 + \tfrac{1}{2}\|W(x' - x_{\texttt{origin}})\|_2^2 \\ \text{s.t.} \quad & c(x'; x_{\texttt{origin}}) = 1 \quad \forall c \in \mathcal{A}, \\ & x' \in \mathcal{F}(\mathcal{C}_{\text{struct}}; x_{\texttt{origin}}), \\ & x'_i = x_i \quad \forall i \notin S, \end{aligned}$$

where W normalizes features by their ranges, $\mathcal{A}$ is the constraint group currently targeted by cyclic projection, and S contains the features allowed to change. The first term limits disruption of the attack candidate; the second discourages drift from the original example.

After each repair, we recheck all learned constraints. Previously satisfied relations that are now violated are added to the local problem. If satisfying them requires changing additional features, we expand S and repair the interacting relations jointly. Cycling through the remaining violations exploits sparse feature dependencies while accounting for their interactions. The cycle terminates when a full pass fails to reduce the number of violated constraints.

Example, step 4: Project and include an interacting constraint. Attack $v = (6, 10, 4, 0.2, 3)$ violates c_R, c_V. Repair $S = \{O, T, r\}$, fixing $C = 4, V = 3$:

$$(6, 10, 0.2) \xrightarrow{c_R} (2, 10, 0.2) \xrightarrow{\{c_R, c_P\}} (2, 6, 1/3) = p_0|_S.$$

First repair breaks c_P, so it joins the local problem. Joint repair restores feasibility but loses misclassification.

[1] Supplying all previously accepted constraints caused the growing constraint history to dominate the context and progressively reduce constraint diversity.

LLM-assisted refinement of constraint formulation. Mathematically equivalent representations can differ in solver efficiency and, for fast numerical solvers such as Couenne (Belotti et al. (2009)), stability. For instance, for scalar values a, b, t with $b \neq 0$, the relation $a/b = t$ can be expressed as $a = bt$, avoiding explicit division by a potentially small denominator that might hinder our solver. In general, constraint reformulation can improve solver performance while preserving the original problem's semantics (Liberti et al., 2009; Bjørner & Fazekas, 2023). When ordinary repair fails, the LLM proposes alternative formulations for the solver, while the original executable constraints remain unchanged and are used to validate proposed repairs. After a fixed number of unsuccessful reformulation attempts, we query the LLM once for values that satisfy the active constraints with minimal change. A proposed candidate repair is accepted only if it satisfies the original active predicates and structural constraints, then continues through the cyclic procedure.

Example, step 5: Assist failed solver call. If c_R fails near $T = 0$ (instability), the LLM reformulates:

$$\begin{cases} O = r = 0, & T = 0, \\ r = O/T, & T > 0 \end{cases} \xrightarrow{\text{LLM}} O = rT, \quad r \leq T.$$

Here, $r \leq T$ enforces $r = 0$ when $T = 0$.

Recovery of misclassification. Recovery is triggered when projection moves an attack candidate into the non-adversarial region (We invoke recovery only in this case because it is substantially slower on average than ordinary projection). Recovery is a neighborhood search for candidates that remain adversarial after repair. CHAT samples Gaussian perturbations over the mutable features to obtain candidates q_i, retains those that remain adversarial, and computes their local projection p_i. Samples already satisfying the constraint group $\mathcal{A}$ and perturbation budget bypass repair. If no repair succeeds, the next search is centered on the adversarial sample with the smallest repair distance $\|W(q_i - p_i)\|_2$, with increased sampling spread. A repaired point that preserves misclassification, satisfies the active relations, and remains within the perturbation budget resumes the cyclic procedure. Exploration continues until recovery succeeds or its budget is exhausted. Recovery uses only predicted labels and therefore applies under both threat models.

Example, step 6: Recover, reanchor, and succeed locally. With $C = 4, V = 3$, recall that repair enforces $T = O + C$ and $r = O/T$, thus we show only (O, T, r). We sample several perturbations. For the first sample, q_1:

$$q_1|_S = (5, 9, 0.4) \longmapsto p_1|_S = (3, 7, 3/7).$$

The sample is adversarial; its repair is not. Its repair distance $\|W(q_1 - p_1)\|_2 \simeq 0.144$ is smaller than the earlier $\|W(v - p_0)\|_2 \simeq 0.313$. If this is the best failed pair (q_i, p_i), re-center the search at q_1 and sample from this new center. From it, sample and repair the same coordinates. Assume we sampled:

$$q_2|_S = (5, 9, 0.6) \longmapsto p_2|_S = (4, 8, 0.5).$$

Now p_2 is adversarial and satisfies c_R, c_P. Cyclic projection continues: c_V still fails because $O = 4 > V = 3$.

Constraint-informed restarts. Among constraints with failed recovery (i.e. exhausted recovery budget), we select the one whose repairs caused the largest cumulative increase in distance from the original example. We freeze its eligible feature (to the projection result that lost misclassification) contributing most to that distance during subsequent base-attack steps. We then continue the search from the point that lost misclassification, with additional steps of the underlying base attack, holding that feature fixed and retaining earlier freezes. This encourages the attack to find candidates through alternative feature changes, while keeping the perturbation budget anchored to $x_{\texttt{origin}}$.

Example, step 7: Assign blame, restart, and accept. Repair c_V, now changing only (O, V). Including c_R, c_P forces $O = 4$, giving $p_3|_{\{O,V\}} = (4, 4)$. [a] Every feasible local repair has $2O - V \leq 4$: misclassification cannot be locally found, thus recovery exhausts. Blame c_V and freeze its larger contributor, $V = 4$, in the base attack. Another run of the base attack from p_3, with $V = 4$ fixed, can yield v^+, which is then projected to p^+:

$$v^+ = (5, 10, 5, 0.5, 4) \longmapsto p^+ = (5, 10, 5, 0.5, 5).$$

Projection may change the attack-frozen V. The final row satisfies all three constraints, remains adversarial ($2O - V = 5$), and lies within the original budget ($D \simeq 0.474 < 0.5$).

[a] Although c_R, c_P constraints are not included initially, subsequent steps will violate them, requiring the projection to account for both, thus restricting O.

We repeat the perturbation–projection cycle until a valid adversarial example is found, the perturbation stage fails to produce an attack candidate, no further constraint-informed restart is possible, or the limit of 20 cycles is reached.

The complete projection and recovery algorithm is included in App. C.3.

4.3 Label-Only CGBA Extensions

For the label-only base attack, we adapt CGBA (see §2) from image classifiers by enforcing tabular structural constraints. CGBA searches a local semi-circle to reduce the distance between the current adversarial boundary point and the clean input. However, we found that this search can stall on local optima when it fails to find a closer adversarial boundary point. We hypothesize that two properties of tabular data make local minima stalling likely: low effective dimensionality limits alternative search directions, while mixed continuous and integer features make small geometric updates unreliable, since rounding can map distinct proposals to the same point. Our extensions are twofold, and operate within the base attack:

- **Multi-Hop Boundary Walk:** Requiring every perturbation step to reduce distance can prevent exploration of other boundary regions. At each hop, we move outward in a new direction away from the clean input to find an adversarial point, then use binary search toward the clean input to reach another point on the boundary, potentially farther from the clean input. Continuing for several hops allows temporary increases in distance to explore regions that strictly improving steps might miss, while retaining the closest attack candidate found across all searches. The full details are in App. D.3.
- **Multi-Direction Semicircle Search:** Relying on a single estimated normal to the classifier's decision boundary to guide the semicircle search may miss useful directions, especially with few mutable features. Searching additional planes provides alternative routes toward closer adversarial points. Besides CGBA's estimated normal, we try directions based on individual mutable features and random combinations thereof. Each direction is made perpendicular to the line connecting the original example and the current candidate. Full details are in App. D.4 and App. D provides the complete algorithm.

5 Experiments

Setup. We evaluate on three TabularBench datasets—URL, LCLD, and WiDS—using four model architectures: TabTransformer, VIME, RLN, and STG, under standard and adversarial training (Simonetto et al., 2024b). Following TabularBench, we evaluate 500 samples per dataset over three attack seeds. TabularBench provides 14, 9, and 31 golden constraints, respectively, per dataset, all manually created; CHAT produced 124, 62, and 158 constraints, respectively, using the LLM. We follow TabularBench's attack settings (Simonetto et al., 2024b), using scaled L_2 distance with $\epsilon = 0.5$. We measure constrained ASR, which is the fraction of inputs whose final outputs are misclassified, constraint-valid, and within budget, including initially misclassified inputs. We compare against the state-of-the-art attack, CAA, and its components, CAPGD and MOEVA, covering combined, gradient-based, and evolutionary constrained attacks (Simonetto et al., 2024a; 2022). CAA matches or exceeds both components in all 24 settings when averaged over three seeds. We therefore omit their standalone results from the main tables. Attack settings are detailed in App. B.

5.1 Constraint Quality

We first ask about the quality of our mined constraints: can the automatically learned constraints preserve valid data, detect inconsistencies, and uncover useful restrictions overlooked by manual rules? We examine these questions through behavioral comparisons and domain informed judgments.

5.1.1 Completeness and soundness.

Following the empirical evaluation framework of CaFA (Ben-Tov et al., 2024), we assess two complementary evaluation components: **completeness** to measure acceptance of held out clean rows, while **soundness** measures rejection of generated golden constraint violations. We use 1,000 clean

rows per dataset for completeness and generate soundness samples from 100 separate seed rows. App. B.3 details how violating samples are generated.

We compare $\mathcal{C}_{\text{LLM}}$ to denial constraints (DCs) and conformance constraints (CCs) (described in App. A.2). DCs rule out combinations of feature comparisons that should not occur within or across records (Chu et al., 2013), whereas CCs capture statistical regularities by bounding approximately invariant linear combinations of numerical features (Fariha et al., 2021). Together, these baselines test how effectively logical and statistical rules learned from data capture the golden constraints compared with our method. DCs suffer from exponential mining growth for each new column added, and thus we measure only on LCLD with some feature selection. CC hyper-parameters are selected based on over 500 different combinations, chosen by the best mean over completeness and soundness.

Table 1: Completeness and soundness of CHAT's $\mathcal{C}_{\text{LLM}}$ and the two constraint-mining baselines across the three datasets. Pooled soundness weighs each generated invalid sample equally; per-target soundness weighs each golden relation yielding generated invalid samples equally. We show several DC constraint counts to control completeness and soundness. Full results appear in App. B.9.

			Evaluated rows			Soundness	
Dataset	Method	#Constr.	Invalid	Clean	Compl.	Pooled	Per-target
LCLD	CHAT's $\mathcal{C}_{\text{LLM}}$	62	2,475	1,000	0.999	1.000	1.000
	Denial constraints	1,000			0.914	0.226	0.239
		5,000			0.662	0.560	0.606
		10,000			0.453	0.699	0.738
	Conformance constraints	180			0.734	0.615	0.667
URL	CHAT's $\mathcal{C}_{\text{LLM}}$	124	829	1,000	0.997	0.975	0.699
	Conformance constraints	674			0.930	0.876	0.922
WiDS	CHAT's $\mathcal{C}_{\text{LLM}}$	158	7,384	1,000	0.984	0.999	0.999
	Conformance constraints	4,843			0.723	0.637	0.675

The results in Table 1 show that CHAT combines high clean-row acceptance (98.4–99.9%) with high violation detection (97.5–100%). Across almost all tested golden constraints, the LLM-generated constraints detect nearly all generated violations. The evaluated DC and CC configurations exhibit a stronger trade-off between these objectives. However, this detection advantage has two exceptions in URL with lower per constraint soundness (indicated by per-target soundness) than CC's. This was due to the LLM failing to capture two hard-to-violate constraints.[2]

In the reverse direction, we test whether golden constraints identify violations of learned constraints. Table 6 (App. B.4) shows that violations of learned constraints were missed by the golden constraints. The golden constraints reject 88.5%, 11.5%, and 66.4% of these generated samples on LCLD, URL, and WiDS, respectively. This showcases that the LLM produces constraints not captured by the golden constraints.

5.1.2 Professionals and LLM validity check.

To assess constraint correctness, we ask domain professionals and two LLMs to evaluate 20 sampled learned constraints and golden constraints for each dataset. These judgments provide qualitative evidence of their domain validity.

Table 2: One professional per domain and average of two LLM judges true rates for learned and golden constraints. Full professional and LLM summary results appear in App. B.5.

	Professional judge				LLM judge			
	Learned		Golden		Learned		Golden	
Dataset	n	True rate	n	True rate	n	True rate	n	True rate
LCLD	20	0.950	9	1.000	20	0.800	9	0.944
URL	20	0.950	10	0.900	20	0.925	14	0.750
WiDS	20	0.900	10	1.000	20	0.875	31	1.000

[2]The two missed constraints were harder to violate, resulting in fewer invalid samples and explaining the difference between the per-target and pooled URL results.

Table 2 supports the validity of $\mathcal{C}_{\text{LLM}}$. Over the same LLM prompt and human judges, the resulting constraints are almost on par with the golden constraints, and were almost all accepted as true.

5.2 Attacking Under Identical Learned Constraints

This comparison tests how effectively complete attack pipelines optimize using the same learned constraints. Every method receives the same learned constraints and is evaluated on the same set of learned constraints.

Table 3: Constrained success under identical learned constraints. Constrained ASR percentages are averaged over three seeds. All attacks receive $\mathcal{C}_{\text{LLM}}$ as input and the constrained ASR is computed w.r.t. $\mathcal{C}_{\text{LLM}}$.

Dataset	Model	Training	Clean Acc. (%)	CHAT_{WB}	CHAT_{BB}	CAA180
URL	TabTransformer	Adv.	95.4	**52.2** (+14.9)	42.9 (+5.6)	37.3
	TabTransformer	Std	94.6	91.5 (+13.3)	**93.4** (+15.2)	78.2
	VIME	Adv.	94.8	25.0 (+0.9)	**28.7** (+4.7)	24.1
	VIME	Std	94.0	**43.6** (+8.6)	40.3 (+5.3)	35.0
	RLN	Adv.	96.8	**50.8** (+12.3)	41.4 (+2.9)	38.5
	RLN	Std	96.0	**90.5** (+19.3)	79.3 (+8.1)	71.2
	STG	Adv.	95.4	**8.9** (+0.7)	8.9 (+0.6)	8.3
	STG	Std	94.6	**48.4** (+15.7)	39.5 (+6.9)	32.7
LCLD	TabTransformer	Adv.	74.4	30.2 (+3.6)	**31.1** (+4.5)	26.6
	TabTransformer	Std	69.6	**96.1** (+14.3)	**96.1** (+14.3)	81.8
	VIME	Adv.	63.2	**90.6** (+10.2)	89.4 (+9.0)	80.4
	VIME	Std	67.0	87.1 (+1.7)	**93.1** (+7.7)	85.4
	RLN	Adv.	68.6	**38.1** (+4.9)	34.2 (+1.0)	33.2
	RLN	Std	69.4	**96.4** (+6.2)	95.4 (+5.2)	90.2
	STG	Adv.	14.6	80.8 (+3.0)	**80.9** (+3.1)	77.8
	STG	Std	62.8	49.1 (+7.3)	**49.8** (+7.9)	41.9
WiDS	TabTransformer	Adv.	77.4	**36.7** (+5.9)	35.1 (+4.3)	30.8
	TabTransformer	Std	75.0	**58.5** (+17.5)	56.3 (+15.3)	41.1
	VIME	Adv.	72.4	**49.8** (+9.4)	46.2 (+5.8)	40.4
	VIME	Std	73.0	**50.8** (+10.3)	48.4 (+7.9)	40.5
	RLN	Adv.	78.4	**32.8** (+1.7)	32.3 (+1.2)	31.1
	RLN	Std	77.6	**38.5** (+5.5)	35.1 (+2.0)	33.1
	STG	Adv.	62.8	**56.8** (+8.1)	52.7 (+4.1)	48.7
	STG	Std	78.0	**38.8** (+5.7)	34.0 (+0.9)	33.1

Table 3 shows that both variants outperform CAA180 under learned constraints and under comparable compute budget (Appendix B.7). Despite using only predicted labels, CHAT_{BB} exceeds white-box CAA180 in all 24 settings. Across all 24 settings, their average gains are 6.0 and 8.4 percentage points, respectively.

Although seemingly counterintuitive, the occasional black-box advantage over our white-box method observed here may be explained by the fact that the two CHAT variants share the same projection and recovery procedures and differ only in the base attack. We hypothesize that the larger perturbations sometimes produced by the black-box base attack help preserve misclassification during projection and facilitate recovery, although further investigation is needed to confirm this explanation.

Computational and query cost. Under shared learned constraints, CHAT_{BB} uses less recorded CPU time per attacked row than the CAA180 reference on all three datasets. It requires tens of thousands to around 10^5 classifier queries per row, while attack-time LLM costs remain below \$0.001 per row. Full results are reported in App. B.8.

5.3 Attack Performance w.r.t. Golden Constraints

The previous subsection uses the same learned constraints for both attack construction and evaluation. Here, we test whether CHAT remains competitive when evaluated against golden constraints unavailable to it during attack construction. CHAT receives only learned constraints, whereas regular CAA receives the golden constraints. All returned examples are evaluated against those golden constraints under the same perturbation budget.

Table 4: Constrained ASR, averaged over three seeds. CHAT uses $\mathcal{C}_{\text{LLM}}$ while CAA uses golden constraints (with its default attack budget). All outputs are evaluated against golden constraints.

Dataset	Model	Training	Clean Acc. (%)	CHAT_{WB}	CHAT_{BB}	Regular CAA
URL	TabTransformer	Adv.	95.4	**54.0** (+11.9)	43.3 (+1.2)	42.1
	TabTransformer	Std	94.6	92.8 (+1.2)	**93.4** (+1.8)	91.6
	VIME	Adv.	94.8	25.3 (-1.7)	**29.1** (+2.1)	27.0
	VIME	Std	94.0	44.6 (-3.1)	41.9 (-5.7)	**47.7**
	RLN	Adv.	96.8	**54.9** (+12.1)	44.7 (+1.8)	42.9
	RLN	Std	96.0	**90.9** (+2.7)	90.4 (+2.3)	88.1
	STG	Adv.	95.4	**9.3** (+0.7)	8.9 (+0.3)	8.6
	STG	Std	94.6	**50.8** (+11.2)	40.0 (+0.4)	39.6
LCLD	TabTransformer	Adv.	74.4	32.6 (+0.4)	**33.5** (+1.3)	32.2
	TabTransformer	Std	69.6	**98.9** (+5.5)	98.9 (+5.5)	93.4
	VIME	Adv.	63.2	**94.5** (+1.5)	93.2 (+0.2)	93.0
	VIME	Std	67.0	90.7 (-8.7)	96.7 (-2.8)	**99.5**
	RLN	Adv.	68.6	**40.7** (+0.6)	36.8 (-3.3)	40.1
	RLN	Std	69.4	99.7 (-0.3)	98.4 (-1.6)	**100.0**
	STG	Adv.	14.6	88.3 (-0.1)	88.4 (-0.1)	**88.5**
	STG	Std	62.8	52.6 (+0.8)	**53.2** (+1.4)	51.8
WiDS	TabTransformer	Adv.	77.4	**36.7** (+2.1)	35.1 (+0.5)	34.5
	TabTransformer	Std	75.0	**58.5** (+4.6)	56.3 (+2.3)	53.9
	VIME	Adv.	72.4	**49.7** (+2.5)	46.2 (-1.0)	47.2
	VIME	Std	73.0	**50.8** (+1.9)	48.4 (-0.5)	48.9
	RLN	Adv.	78.4	**32.8** (+0.3)	32.3 (-0.1)	32.5
	RLN	Std	77.6	**38.5** (+1.4)	35.1 (-2.1)	37.1
	STG	Adv.	62.8	**56.8** (+3.5)	52.7 (-0.5)	53.3
	STG	Std	78.0	**38.8** (+3.0)	34.0 (-1.8)	35.8

Table 4 shows that CHAT's variants achieve performance comparable to CAA given the golden constraints, despite having no access to those constraints during attack.

6 Conclusion and Limitations

We introduced CHAT, to our knowledge the first label-only tabular attack that generates adversarial examples satisfying complex relational constraints. Our attacks remain competitive with white-box CAA in constrained success rate (CSR), even when only CAA receives the benchmark's golden constraints and all outputs are judged against them. This holds across datasets and architectures, with and without adversarial training.

CHAT uses an LLM to learn relationships between features from their descriptions and unlabeled data. Domain-expert assessments and empirical validation support the rules' validity and suggest that they largely capture the tested golden constraints while identifying additional domain relationships not in the golden constraints. When all attacks receive and are evaluated against the same auto-generated constraints, CHAT achieves higher CSR than the evaluated white-box CAA baseline in all settings.

Whereas attacks such as CAA and MOEVA jointly optimize misclassification and constraint satisfaction, CHAT adapts unconstrained base attacks to complex relational constraints through a projection and misclassification recovery framework.

Our main limitations concern constraint coverage and computational cost. The LLM may miss rules used in evaluation, and covering more comprehensive constraint sets may require additional mining iterations. Computational and query costs also limit use with rate-limited services and in adversarial training, where the method's reliance on LLM assistance for difficult constraint repairs can become expensive. Future work should therefore aim to adapt attacks to complex constraints with reduced LLM assistance in the attack loop and reduce classifier queries. Another future work direction is to study how different base attacks interact with our constraint repair. Understanding this interaction could help explain why our label-only variant sometimes outperforms its white-box counterpart and guide the design of stronger constrained repair based attacks. Building on this understanding, future work could also explore adaptive selection or combination of base attacks, evaluate the resulting attacks against stateful defenses, and investigate their use in adversarial training.

AI USE STATEMENT

We used generative AI tools to assist with literature review, software implementation and debugging, hyperparameter tuning, code for visualizing experimental results, and manuscript drafting and polishing. The research ideas and experimental design were developed by the authors, who evaluated the results and drew the scientific conclusions. As part of CHAT, LLMs were used to propose and refine relational constraints and assist with constraint reformulation and candidate repair, as described in the method section. The authors reviewed the AI-assisted contributions and take responsibility for the final manuscript, implementation, and reported findings.

REFERENCES

Ashlesha Akella, Akshar Kaul, Krishnasuri Narayanam, and Sameep Mehta. 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*, pp. 2713–2748, Suzhou, China, 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.emnlp-industry.183. URL https://aclanthology.org/2025.emnlp-industry.183/.

Nour Alhussien, Gagan Agrawal, and Ahmed Aleroud. Augmented tabular adversarial evasion attacks with constraint satisfaction guarantees. In *Availability, Reliability and Security*, volume 15993 of *Lecture Notes in Computer Science*, pp. 365–386. Springer Nature Switzerland, 2025. doi: 10.1007/978-3-032-00627-1_18. URL https://doi.org/10.1007/978-3-032-00627-1_18.

Pietro Belotti, Jon Lee, Leo Liberti, François Margot, and Andreas Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4–5):597–634, 2009. doi: 10.1080/10556780903087124. URL https://optimization-online.org/2008/08/2059/.

Matan Ben-Tov, Daniel Deutch, Nave Frost, and Mahmood Sharif. CaFA: Cost-aware, feasible attacks with database constraints against neural tabular classifiers. In *2024 IEEE Symposium on Security and Privacy*, pp. 1345–1364. IEEE, 2024. doi: 10.1109/SP54263.2024.00218. URL https://arxiv.org/abs/2501.10013.

Nikolaj Bjørner and Katalin Fazekas. On incremental pre-processing for SMT. In *Automated Deduction – CADE 29*, volume 14132 of *Lecture Notes in Computer Science*, pp. 41–60. Springer, 2023. doi: 10.1007/978-3-031-38499-8_3. URL https://kfazekas.github.io/papers/BjornerFazekas-CADE23.pdf.

Hamid Bostani, Zhengyu Zhao, Zhuoran Liu, and Veelasha Moonsamy. Level up with ML vulnerability identification: Leveraging domain constraints in feature space for robust Android malware detection. *ACM Transactions on Privacy and Security*, 28(2):1–32, 2025. doi: 10.1145/3711899. URL https://arxiv.org/abs/2205.15128.

Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1712.04248.

Francesco Cartella, Orlando Anunciação, Yuki Funabiki, Daisuke Yamaguchi, Toru Akishita, and Olivier Elshocht. Adversarial attacks for tabular data: Application to fraud detection and imbalanced data. In *Proceedings of the Workshop on Artificial Intelligence Safety 2021 (SafeAI 2021)*, volume 2808 of *CEUR Workshop Proceedings*, 2021. URL https://ceur-ws.org/Vol-2808/Paper_4.pdf.

Jianbo Chen, Michael I. Jordan, and Martin J. Wainwright. HopSkipJumpAttack: A query-efficient decision-based attack. In *2020 IEEE Symposium on Security and Privacy*, pp. 1277–1294. IEEE, 2020. doi: 10.1109/SP40000.2020.00045. URL https://arxiv.org/abs/1904.02144.

Alesia Chernikova and Alina Oprea. FENCE: Feasible evasion attacks on neural networks in constrained environments. *ACM Transactions on Privacy and Security*, 25(4):1–34, 2022. doi: 10.1145/3544746. URL https://arxiv.org/abs/1909.10480.

Xu Chu, Ihab F. Ilyas, and Paolo Papotti. Discovering denial constraints. *Proceedings of the VLDB Endowment*, 6(13):1498–1509, 2013. doi: 10.14778/2536258.2536262. URL https://www.vldb.org/pvldb/vol6/p1498-papotti.pdf.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pp. 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24. URL https://www.microsoft.com/en-us/research/publication/z3-an-efficient-smt-solver/.

Kevin Eykholt, Taesung Lee, Douglas Schales, Jiyong Jang, Ian Molloy, and Masha Zorin. URET: Universal robustness evaluation toolkit (for evasion). In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 3817–3833. USENIX Association, 2023. URL https://www.usenix.org/conference/usenixsecurity23/presentation/eykholt.

Anna Fariha, Ashish Tiwari, Arjun Radhakrishna, Sumit Gulwani, and Alexandra Meliou. Conformance constraint discovery: Measuring trust in data-driven systems. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 499–512. Association for Computing Machinery, 2021. doi: 10.1145/3448016.3452795. URL https://arxiv.org/abs/2003.01289.

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015. URL https://arxiv.org/abs/1412.6572.

William E. Hart, Jean-Paul Watson, and David L. Woodruff. Pyomo: Modeling and solving mathematical programs in python. *Mathematical Programming Computation*, 3(3):219–260, 2011. doi: 10.1007/s12532-011-0026-8. URL https://link.springer.com/article/10.1007/s12532-011-0026-8.

Klim Kireev, Bogdan Kulynych, and Carmela Troncoso. Adversarial robustness for tabular data through cost and utility awareness. In *Network and Distributed System Security Symposium (NDSS)*, 2023. doi: 10.14722/ndss.2023.24924. URL https://www.ndss-symposium.org/wp-content/uploads/2023-924-paper.pdf.

Leo Liberti, Sonia Cafieri, and Fabien Tarissan. Reformulations in mathematical programming: A computational approach. In *Foundations of Computational Intelligence, Volume 3: Global Optimization*, volume 203 of *Studies in Computational Intelligence*, pp. 153–234. Springer, 2009. doi: 10.1007/978-3-642-01085-9_7. URL https://www.lix.polytechnique.fr/~liberti/arschapter.pdf.

Ester Livshits, Alireza Heidari, Ihab F. Ilyas, and Benny Kimelfeld. Approximate denial constraints. *Proceedings of the VLDB Endowment*, 13(10):1682–1695, 2020. doi: 10.14778/3401960.3401966. URL https://www.vldb.org/pvldb/vol13/p1682-livshits.pdf.

Yael Mathov, Eden Levy, Ziv Katzir, Asaf Shabtai, and Yuval Elovici. Not all datasets are born equal: On heterogeneous tabular data and adversarial examples. *Knowledge-Based Systems*, 242:108377, 2022. doi: 10.1016/j.knosys.2022.108377. URL https://arxiv.org/abs/2010.03180.

Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. Intriguing properties of adversarial ML attacks in the problem space. In *IEEE symposium on security and privacy (S&P)*, pp. 1332–1349. IEEE, 2020. doi: 10.1109/SP40000.2020.00073. URL https://discovery.ucl.ac.uk/id/eprint/10132359/.

Md Farhamdur Reza, Ali Rahmati, Tianfu Wu, and Huaiyu Dai. CGBA: Curvature-aware geometric black-box attack. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 124–133, 2023. doi: 10.1109/ICCV51070.2023.00018. URL https://openaccess.thecvf.com/content/ICCV2023/html/Reza_CGBA_Curvature-aware_Geometric_Black-box_Attack_ICCV_2023_paper.html.

Ryan Sheatsley, Blaine Hoak, Eric Pauley, Yohan Beugin, Michael J. Weisman, and Patrick McDaniel. On the robustness of domain constraints. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 495–515. Association for Computing Machinery, 2021. doi: 10.1145/3460120.3484570. URL https://arxiv.org/abs/2105.08619.

Thibault Simonetto, Salijona Dyrmishi, Salah Ghamizi, Maxime Cordy, and Yves Le Traon. A unified framework for adversarial attack and defense in constrained feature space. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 1313–1319. International Joint Conferences on Artificial Intelligence Organization, 2022. doi: 10.24963/ijcai.2022/183. URL https://www.ijcai.org/proceedings/2022/183.

Thibault Simonetto, Salah Ghamizi, and Maxime Cordy. Constrained adaptive attack: Effective adversarial attack against deep neural networks for tabular data. In *Advances in Neural Information Processing Systems*, volume 37, 2024a. doi: 10.52202/079017-0873. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/3106a1387b24c33bf66682e7ff393d56-Abstract-Conference.html.

Thibault Simonetto, Salah Ghamizi, and Maxime Cordy. TabularBench: Benchmarking adversarial robustness for tabular deep learning in real-world use-cases. In *Advances in Neural Information Processing Systems*, volume 37, 2024b. doi: 10.52202/079017-2492. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/8f5defaa8dc728c3e4aaee59ac8ca942-Abstract-Datasets_and_Benchmarks_Track.html.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014. URL https://arxiv.org/abs/1312.6199.

A Additional Related Work

Table 5: Comparing tabular attacks according to the settings they are applicable to (WB for white-box, BB-score for score-based black-box, and BB-label for hard-label black-box), the constraint types they cover (logical and/or arithmetic relations), and whether constraints are automatically produced.

Method	Setting			Constraints		
	WB	BB-score	BB-label	Logical	Arithmetic	Automatic
CaFA (Ben-Tov et al., 2024)	●	○	○	●	○	●
CAA / CAPGD (Simonetto et al., 2024a)	●	○	○	●	●	○
CSP (Sheatsley et al., 2021)	●	○	○	●	○	●
MOEVA (Simonetto et al., 2022)	●	●	○	●	●	○
URET (Eykholt et al., 2023)	●	○	○	○	◐	○
Bostani et al. (2025)	●	○	○	◐	○	●
FENCE (Chernikova & Oprea, 2022)	●	○	○	○	●	○
CHAT (ours)	●	●	●	●	●	●

● supported/reported; ◐ partial/conditional; ○ unsupported/not reported.

A.1 Constrained Tabular Attacks

Small perturbations do not necessarily produce valid tabular samples; attacks must also respect immutable attributes, feature domains, and relations among features. Examples include changing a loan amount without adjusting its installment at a fixed interest rate and term, or changing a patient's weight without updating the corresponding body mass index. MOEVA addresses these requirements through multiobjective evolutionary search over misclassification, distance, and constraint satisfaction (Simonetto et al., 2022). CAPGD incorporates a differentiable constraint-violation penalty into gradient-based search. CAA combines the two, applying MOEVA where CAPGD fails (Simonetto et al., 2024a), and serves as the primary attack in TabularBench (Simonetto et al., 2024b) and our primary comparison.

CaFA addresses the setting where relational constraints must be learned from data (Ben-Tov et al., 2024). It combines TabPGD with automatically mined approximate denial constraints and Z3-based projection, using repair heuristics that help preserve misclassification. It therefore establishes a precedent for attacks that acquire and enforce constraints without manually specified relational rules.

Additional approaches incorporate feature dependencies in different ways: CSP learns Boolean constraints (Sheatsley et al., 2021), while Bostani et al. learn statistical feature dependencies (Bostani

et al., 2025). URET and FENCE preserve input validity through supplied transformations and dependency-update procedures (Eykholt et al., 2023; Chernikova & Oprea, 2022). Table 5 summarizes the capabilities and reported evaluation protocols of these and other selected constrained attacks. TCGE (Alhussien et al., 2025) enforces constraints through rollback, but we omit it from our comparison table because its label-only evaluation lacks a prescribed norm bound. Here, "realistic adversarial examples" is not satisfied since not norm bound is present in the eval. It must be misclassified, constraint-valid, and within that bound, yet this for doesn't meet those requirements to full empirically tested black-box adversarial attack. Thus, to our knowledge, ours is the first hard-label black-box attack evaluated under these joint requirements with complex relational constraints. Other prior work on black-box tabular attacks considers feature editability and imperceptibility in fraud detection, attack costs and utility, and data consistency through learned representations and surrogate transfer (Cartella et al., 2021; Kireev et al., 2023; Mathov et al., 2022).

CHAT follows the auto generated setting, differing in its use of LLM proposed executable validators and approximate solver based repair. Its contribution concerns how constraints are acquired and enforced, even when evaluated against unknown constraints, with support for both gradient based and label-only attacks.

A.2 Constraint Discovery for Tabular Data

Discovering constraints from a valid record set is difficult because without invalid examples, observed regularities alone may not distinguish genuine validity requirements from incidental patterns. One approach searches for denial constraints, which rule out combinations of comparisons within or across records and can express integrity rules such as keys and functional dependencies (Chu et al., 2013). Approximate denial constraints extend this by permitting a bounded fraction of violations, allowing such rules to be discovered in noisy data (Livshits et al., 2020). Another approach we consider is conformance constraints, which capture statistical regularities among numerical attributes through a PCA-based discovery procedure that identifies linear combinations with low variance and bounds their variation to obtain approximately invariant relations (Fariha et al., 2021).

LLMs provide another approach by proposing rules informed by feature semantics. Akella et al. (2025) combine LLM generated data quality rules with code generation to produce executable Python validators for detecting data errors. CHAT applies a similar executable rule approach to adversarial attacks, using empirically checked validators to define and enforce a learned feasible region.

A.3 Constraint Projection and Optimization Solvers

Adversarial search may produce candidates that violate feature relations, motivating repair procedures that restore feasibility while preserving the attack's success. Repairing an adversarial candidate requires satisfying constraints while limiting changes that may undo misclassification. CaFA uses Z3, a satisfiability modulo theories solver supporting logical and arithmetic formulas, to repair candidates under mined denial constraints (Ben-Tov et al., 2024; de Moura & Bjørner, 2008). Distance based repair seeks a nearby row satisfying the constraints.

CHAT uses Couenne (through Pyomo package) to formulate distance based repair under learned constraints and mixed feature domains, with extensions to conditional, disjunctive and mixed integer relations, and then approximately solve the resulting nonconvex mixed-integer nonlinear programs (Hart et al., 2011) (Belotti et al., 2009). While using Z3 can give exact solution, it cannot give approximate optimization solution in our attempts, despite documentation saying otherwise. Thus the approximate nature of the Couenne allows it to give non optimal but close solutions, while being much faster and more stable (at least by our attempts, we cannot fully confirm this statement).

CHAT uses these tools for distance-minimizing repair through cyclic approximate solves over small feature subsets. This is a modeling and decomposition choice: classical convergence results for projections onto convex sets do not directly extend to its nonconvex learned feasible regions. We therefore evaluate the procedure empirically without claiming guaranteed convergence.

A.4 Decision-Based Black-Box Attacks

The field of full black-box adversarial attack (i.e. predicted labels only) on image classifier models has been progressing for the last several years. Boundary Attack introduced this setting by starting from initially far adversarial examples, iteratively moving an adversarial point toward the clean input while retaining only adversarial proposals (Brendel et al., 2018). One of the major results found is best white-box results can be matched with enough samples - And thus the subsequent work focused on reducing query cost.

HopSkipJump improved the query efficiency through using boundary search and adding randomized normal estimation (Chen et al., 2020) for better step progression. CGBA is one of the most efficient attacks to date, traversing the boundary line (between the correct and the incorrect label). Using search along the estimated boundary normal with a curvature aware search along semicircular paths in a two dimensional plane (Reza et al., 2023).

We instantiate CHAT under label-only access by adapting CGBA to adhere to structural constraints, with the rest of our method extending to relational constraints. We additionally introduce a multi-hop boundary extending walk that explores alternative boundary regions when local traversal stalls or conflicts with the feasible region.

B Experimental Setup and Supporting Experiments

B.1 Cohort selection, model grid, constraints, and attack settings.

Cohort and model grid. The main suite uses three datasets, four architectures, two training regimes, eight attack configurations, and seeds $0, 1, 2$, giving 576 runs. For each dataset, the evaluator selects class-1 rows satisfying the TabularBench golden relations, draws 600 rows with selection seed 42, and removes the first 100 used by other ablations studies, reserving the next 500 for the main evaluation. Initial correctness on the leaned constraints is not a selection criterion for the cohort. Golden relations are used by the evaluator for this selection but are not supplied to CHAT. The auxiliary studies below use separately selected 100-row cohorts, with their selection settings specified in each subsection.

Constraints and attack settings. Learned attacks use judged constraint sets containing 62 LCLD, 124 URL, and 158 WiDS validators. These were collected by querying the LLM for k=2 constraints per feature. We did not increase k because additional passes yielded mostly duplicate constraints on the tested datasets.[3] These retain candidates accepted as mined by the domain-realism judge, followed by de-duplication. The suite admits records with both translations available, without additional constraint-ID, gradient-usability, or clean-cohort pass-rate filtering. After generation is complete, we run the attack on 100 held-out samples to help eliminate translation errors. The resulting translations are saved for subsequent attacks. However, reformulations produced during any attack evaluated in this paper are not saved for reuse by other attacks. All attacks preserve immutable and categorical features and use scaled L_2 with $\epsilon = 0.5$. Final golden and learned evaluation uses 10^{-6} as tolerance. In the white-box setting, we can generate initial candidates using either TabPGD (Ben-Tov et al. (2024)) or CAA (Simonetto et al. (2024a)), both without relational constraints. For all current results cited in this paper, CAA was the starting base attack used, and for subsequent restarts, adapted TabPGD was used (both without any relational constraints).

B.2 Constraint Translation, Validation and Quality

Evaluation representations. TabularBench requires constraints in its own representation, so we translate the learned Python validators and check agreement on held-out clean rows and constraint-specific invalid probes. These tests provide empirical evidence of agreement, not proof of equivalence over the full domain. However, the identical constraint attack table reports a single learned CSR when source and translated rates agree to three decimal places. All currently reported rates agree at this precision.

An LLM translates each c from Python into a constraint $\widetilde{c}$ to be fed to the Couenne solver Belotti et al. (2009), formalized in the solver's specification language. The Python validator remains a

[3] More complex datasets may require additional passes.

directly testable reference while translations are reformulated to suit the solver, e.g., $AB = C$ is more numerically stable versus $A = C/B$ when $B \neq 0$ that might be the intended constraint. We accept the translation only when the two agree on both held out clean rows and constraint specific invalid rows:

$$c(z) = \widetilde{c}(z), \qquad \forall z \in \mathcal{D}_{\text{hold}} \cup \mathcal{D}^{c}_{\text{invalid}}.$$

We construct $\mathcal{D}^{c}_{\text{invalid}}$ by perturbing clean rows within feature bounds and types, retaining only rows rejected by c (App. C.2).

Realism audit. Construction used OpenAI's o3 (`o3-2025-04-16`) for constraint generation and the initial realism filter. This is due to o3 training data predating the publish date of tabularbench, and thus we can limit a situation where the model produces constraints it has seen in training. Only candidates initially accepted as mined were retained. Learned-constraint assessment replayed the recorded construction-judge prompts.

B.3 Constraint-Quality Sample Generation

Starting from clean rows, we modify mutable numerical features using random replacements and searches toward their bounds, retaining changes that violate the targeted golden constraint. If a search finds no violating row, no sample is generated, so sample counts vary widely across constraints. Pooled soundness weights generated invalid samples equally; per-target soundness weights each constraint yielding samples equally, regardless of its sample count.

DCs however fail to capture some constraints such as (`ratio_open_acc_total_acc == open_acc/total_acc`); they reject only 72.9% of 898 invalid samples generated for that specific constraint (in the best case).

B.4 Coverage Beyond Golden Constraints Full Results

We test whether learned constraints identify violations accepted by the golden rules by generating samples that violate learned constraints and evaluating them against the golden set.

Table 6: Golden-constraint rejection of generated invalid samples violating CHAT-mined constraints. Pooled rates weight generated invalid samples equally; per-target rates weight learned constraints yielding generated invalid samples equally.

				Soundness of golden	
Dataset	#Golden	#Learned	Invalid rows	Pooled	Per-target
LCLD	9	62	6,768	0.885	0.744
URL	14	124	22,373	0.115	0.118
WiDS	31	158	16,259	0.664	0.519

B.5 Professional and LLM Validity Judgments

We asked field professionals (i.e. financial/security/medical professionals) and 2 different models to judge true, false or uncertain on each constraint. The table shows true and uncertain rates.

Over the same LLM prompt and human judges - the resulting constraints are almost on par with the golden constraints.

Table 7: Domain realism verdicts from Claude Opus 5 and GPT-5.6-sol and 3 human professional (each per field) after constraint set random selection of 20 learned constraints.

		LLM judge			Professional judge		
Set	Dataset	LLM n	True rate	Uncertain rate	Human n	True rate	Uncertain rate
Learned (20 sampled)	LCLD	20	0.800	0.025	20	0.950	0.050
Golden (all \ 9)	LCLD	9	0.944	0.000	9	1.000	0.000
Learned (20 sampled)	URL	20	0.925	0.050	20	0.950	0.050
Golden (all \ 10)	URL	14	0.750	0.000	10	0.900	0.100
Learned (20 sampled)	WiDS	20	0.875	0.000	20	0.900	0.100
Golden (all \ 10)	WiDS	31	1.000	0.000	10	1.000	0.000
Learned (20 sampled)	*All*	60	0.867	0.025	60	0.933	0.067
Golden (all \ 29)	*All*	54	0.926	0.000	29	0.966	0.034

B.6 Projection and Recovery Supporting Experiment

We compare progressively more complete configurations: monolithic projection, plain cyclic projection, and full CHAT_{WB} recovery, reporting constraint satisfaction separately from attack success.

Configuration. TabTransformer Std/Madry on LCLD, URL, and WiDS, 100 class-1 test rows per dataset, selection seed 42 and attack seed 0, without filtering by initial correctness. Each dataset–model pair uses one relationship-free CAA initialization with 100 CAPGD steps, 100 MOEVA generations, and 200 offspring; all configurations receive exactly the same saved initial candidates. The full configuration retains native TabPGD for subsequent retries. All runs use CPU execution on eight cores, with the same cohorts, learned constraints, perturbation radius, and solver limits across configurations.

Table 8: Projection/recovery configurations (%), over 100-row cohort. OK is constraint satisfaction, CSR is constrained ASR. Note the Full CHAT runs a different cohort then other runs, thus results differ.

Dataset	Training	Clean acc.	Monolithic		Cyclic		Full CHAT recovery	
			OK	CSR	OK	CSR	OK	CSR
URL	Std	93	14	7	100	32	98	**91**
URL	Madry	91	57	9	100	43	100	**50**
LCLD	Std	75	23	22	86	79	93	**93**
LCLD	Madry	80	79	17	97	26	97	**29**
WiDS	Std	77	59	23	100	**62**	100	**62**
WiDS	Madry	77	78	23	100	**43**	100	**43**

Conclusion. Table 8 shows that full recovery increases CSR over plain cyclic projection in the four URL and LCLD settings and ties it in both WiDS settings. Mean CSR rises from 47.5% to 61.3%. Cyclic projection reaches 100% source validity on URL and WiDS, so validity alone does not establish attack success. Monolithic projection produces no successful attacks on initially correct rows; its nonzero CSR comes from initially misclassified rows whose final outputs pass the validity and budget checks. The full configuration combines local recovery, LLM assistance, and outer retries; this comparison measures their combined effect and is not compute matched. The guide-only configuration matches plain cyclic CSR in all six settings and makes no LLM requests.

B.7 CAA Search-Budget Sensitivity Supporting Experiment

Matching computational budgets across attacks is challenging: CHAT has variable runtime, heavily dependent on how hard individual constraints are to solve at a given moment for our solver, while CAA's search budget does not directly bound execution time. We evaluate multiple CAA configurations and select a reference with higher measured average CPU time than our selected black-box configuration, without claiming exact computational cost comparison.

We vary CAA's CAPGD steps, MOEVA generations, and offspring counts across eight search configurations while holding the production CHAT_{WB} profile fixed. These are CAA-specific search controls, not shared units of compute. All other configuration knobs didn't have a meaningful affect.

Configuration. TabTransformer Std/Madry on LCLD, URL, and WiDS, 100 test rows per dataset. Table 9 reports success on learned constraints when both are given the learned constraints.

Table 9: CAA search-budget sensitivity: S/G/O: CAPGD steps / MOEVA generations / offspring per generation within CAA. Note CHAT runs a different cohort then other runs, thus results differ.

				LCLD		URL		WiDS	
Attack	S	G	O	Std	Madry	Std	Madry	Std	Madry
CAA (default budget)	10	100	100	73.0	22.0	63.0	29.0	42.0	34.0
CAA	100	100	100	73.0	22.0	62.0	24.0	42.0	33.0
CAA180	100	100	180	74.0	22.0	77.0	37.0	40.0	35.0
CAA	300	100	180	74.0	22.0	78.0	39.0	42.0	37.0
CAA	100	200	180	73.0	22.0	74.0	44.0	45.0	37.0
CAA	100	300	180	73.0	22.0	76.0	43.0	46.0	39.0
CAA	100	100	300	71.0	22.0	80.0	44.0	45.0	35.0
CAA	300	200	180	74.0	22.0	77.0	**47.0**	46.0	37.0
CHAT WB	–	–	–	**91.0**	**23.0**	**88.0**	**47.0**	**62.0**	**42.0**

Conclusion. Under these settings CAA180 was the best config that had the only reasonable mean CPU runtime relative to our attack, and thus was selected.

B.8 Computational and Query Cost

We report descriptive online resource accounting using CPU time and recorded row-search label queries per attacked row.

Table 10: Online CPU seconds and label queries per attacked row, averaged over models and seeds. Ratios use learned-constraint CAA180 as the reference. regular CAA uses golden constraints and its smaller default search budget.

Dataset	CPU seconds per attacked row			Relative to CAA180	BB queries	LLM call cost
	CHAT_{BB}	CAA180	Regular CAA	CHAT_{BB}	per row	USD/sample
URL	11.3	80.9	2.3	0.14×	107 341	0.000381
LCLD	8.6	11.8	3.6	0.73×	42 617	0.000290
WiDS	8.4	15.2	3.1	0.55×	73 664	0.000001

B.9 Constraint Quality: Full Results

Table 11 includes both DC-mining variants. The single-run variant mines one fixed set of 11 LCLD features. The subset-union variant combines and deduplicates constraints obtained from mining ten-feature subsets drawn from 13 features - timed-out runs do not contribute. For each variant, we retain the top 1,000, 5,000, or 10,000 constraints from its ranked list. Single run achieves higher completeness at every count, whereas subset union achieves higher soundness at 5,000 and 10,000 constraints.

Table 11: Full constraint-quality results. All methods use the same clean and generated invalid rows within each dataset. Pooled soundness weights invalid rows equally; per-target soundness weights golden constraints yielding invalid rows equally.

			Evaluated rows			Soundness	
Dataset	Method	#Constr.	Invalid	Clean	Compl.	Pooled	Per-target
LCLD	CHAT's $\mathcal{C}_{\text{LLM}}$	62	2,475	1,000	0.999	1.000	1.000
	Denial constraints, single run	1,000			0.914	0.226	0.239
		5,000			0.662	0.560	0.606
		10,000			0.453	0.699	0.738
	Denial constraints, subset union	1,000			0.865	0.217	0.236
		5,000			0.569	0.579	0.612
		10,000			0.333	0.767	0.795
	Conformance constraints	180			0.734	0.615	0.667
URL	CHAT's $\mathcal{C}_{\text{LLM}}$	124	829	1,000	0.997	0.975	0.699
	Conformance constraints	674			0.930	0.876	0.922
WiDS	CHAT's $\mathcal{C}_{\text{LLM}}$	158	7,384	1,000	0.984	0.999	0.999
	Conformance constraints	4,843			0.723	0.637	0.675

B.10 Full Attack Comparisons

The full comparisons for Sections 5.2 and 5.3, including standalone CAPGD and MOEVA, are reported below.

Table 12: Full comparison under identical learned constraints, including CAPGD and MOEVA. Constrained ASR percentages are averaged over 3 seeds on the fixed cohort. A single value means translated and source percentages agree to one decimal; otherwise we show translated\source. Bold marks the highest unrounded translated rate; parentheses give our gap to the best baseline in percentage points. Clean accuracy is a percentage. Adv. denotes adversarial training.

Dataset	Model	Training	Clean Acc. (%)	CHAT_{WB}	CHAT_{BB}	CAA180	CAPGD	MOEVA
URL	TabTransformer	Adv.	95.4	**52.2** (+14.9)	42.9 (+5.6)	37.3	4.6	7.5
	TabTransformer	Std	94.6	91.5 (+13.3)	**93.4** (+15.2)	78.2	5.4	9.7
	VIME	Adv.	94.8	25.0 (+0.9)	**28.7** (+4.7)	24.1	5.2	7.1
	VIME	Std	94.0	**43.6** (+8.6)	40.3 (+5.3)	35.0	6.0	8.4
	RLN	Adv.	96.8	**50.8** (+12.3)	41.4 (+2.9)	38.5	3.2	6.1
	RLN	Std	96.0	**90.5** (+19.3)	79.3 (+8.1)	71.2	4.0	7.3
	STG	Adv.	95.4	**8.9** (+0.7)	8.9 (+0.6)	8.3	4.6	5.2
	STG	Std	94.6	**48.4** (+15.7)	39.5 (+6.9)	32.7	5.4	11.6
LCLD	TabTransformer	Adv.	74.4	30.2 (+3.6)	**31.1** (+4.5)	26.6	23.2	23.3
	TabTransformer	Std	69.6	**96.1** (+14.3)	**96.1** (+14.3)	81.8	27.6	28.6
	VIME	Adv.	63.2	**90.6** (+10.2)	89.4 (+9.0)	80.4	33.0	36.9
	VIME	Std	67.0	87.1 (+1.7)	**93.1** (+7.7)	85.4	29.4	36.1
	RLN	Adv.	68.6	**38.1** (+4.9)	34.2 (+1.0)	33.2	28.8	30.0
	RLN	Std	69.4	**96.4** (+6.2)	95.4 (+5.2)	90.2	27.6	42.9
	STG	Adv.	14.6	80.8 (+3.0)	**80.9** (+3.1)	77.8	77.8	77.8
	STG	Std	62.8	49.1 (+7.3)	**49.8** (+7.9)	41.9	33.8	33.9
WiDS	TabTransformer	Adv.	77.4	**36.7** (+5.9)	35.1 (+4.3)	30.8	22.6	22.6
	TabTransformer	Std	75.0	**58.5** (+17.5)	56.3 (+15.3)	41.1	25.0	25.9
	VIME	Adv.	72.4	**49.8** (+9.4)	46.2 (+5.8)	40.4	27.6	27.8
	VIME	Std	73.0	**50.8** (+10.3)	48.4 (+7.9)	40.5	27.0	27.6
	RLN	Adv.	78.4	**32.8** (+1.7)	32.3 (+1.2)	31.1	21.6	22.1
	RLN	Std	77.6	**38.5** (+5.5)	35.1 (+2.0)	33.1	22.4	22.9
	STG	Adv.	62.8	**56.8** (+8.1)	52.7 (+4.1)	48.7	37.2	37.6
	STG	Std	78.0	**38.8** (+5.7)	34.0 (+0.9)	33.1	22.0	22.7

Table 13: Full end-to-end comparison, including CAPGD and MOEVA. Constrained ASR percentages are averaged over 3 seeds on the fixed cohort. CHAT uses learned constraints; CAA, CAPGD, and MOEVA use golden constraints. All outputs are evaluated against golden constraints. Regular CAA uses its default budget; this is not compute matched. Bold marks the highest unrounded rate; parentheses give our gap to the best baseline in percentage points. Clean accuracy is a percentage. Adv. denotes adversarial training.

Dataset	Model	Training	Clean Acc. (%)	CHAT_{WB}	CHAT_{BB}	Regular CAA	CAPGD	MOEVA
URL	TabTransformer	Adv.	95.4	**54.0** (+11.9)	43.3 (+1.2)	42.1	16.8	41.4
	TabTransformer	Std	94.6	92.8 (+1.2)	**93.4** (+1.8)	91.6	84.4	80.4
	VIME	Adv.	94.8	25.3 (-1.7)	**29.1** (+2.1)	27.0	20.2	26.9
	VIME	Std	94.0	44.6 (-3.1)	41.9 (-5.7)	**47.7**	39.2	38.7
	RLN	Adv.	96.8	**54.9** (+12.1)	44.7 (+1.8)	42.9	17.0	41.8
	RLN	Std	96.0	**90.9** (+2.7)	90.4 (+2.3)	88.1	71.8	74.3
	STG	Adv.	95.4	**9.3** (+0.7)	8.9 (+0.3)	8.6	7.8	8.5
	STG	Std	94.6	**50.8** (+11.2)	40.0 (+0.4)	39.6	25.0	38.3
LCLD	TabTransformer	Adv.	74.4	32.6 (+0.4)	**33.5** (+1.3)	32.2	28.4	25.8
	TabTransformer	Std	69.6	**98.9** (+5.5)	98.9 (+5.5)	93.4	47.6	31.1
	VIME	Adv.	63.2	**94.5** (+1.5)	93.2 (+0.2)	93.0	86.0	39.5
	VIME	Std	67.0	90.7 (-8.7)	96.7 (-2.8)	**99.5**	95.6	36.8
	RLN	Adv.	68.6	**40.7** (+0.6)	36.8 (-3.3)	40.1	37.0	31.9
	RLN	Std	69.4	99.7 (-0.3)	98.4 (-1.6)	**100.0**	96.8	32.9
	STG	Adv.	14.6	88.3 (-0.1)	88.4 (-0.1)	**88.5**	87.8	85.4
	STG	Std	62.8	52.6 (+0.8)	**53.2** (+1.4)	51.8	51.0	37.4
WiDS	TabTransformer	Adv.	77.4	**36.7** (+2.1)	35.1 (+0.5)	34.5	30.2	24.7
	TabTransformer	Std	75.0	**58.5** (+4.6)	56.3 (+2.3)	53.9	43.4	30.5
	VIME	Adv.	72.4	**49.7** (+2.5)	46.2 (-1.0)	47.2	35.8	31.6
	VIME	Std	73.0	**50.8** (+1.9)	48.4 (-0.5)	48.9	42.0	32.5
	RLN	Adv.	78.4	**32.8** (+0.3)	32.3 (-0.1)	32.5	30.4	25.0
	RLN	Std	77.6	**38.5** (+1.4)	35.1 (-2.1)	37.1	31.0	25.1
	STG	Adv.	62.8	**56.8** (+3.5)	52.7 (-0.5)	53.3	38.4	38.9
	STG	Std	78.0	**38.8** (+3.0)	34.0 (-1.8)	35.8	25.6	23.5

C Constraint Generation and Projection

Inputs and generation. For each selected feature i, the LLM receives the descriptions of all features and an example clean row. Descriptions specify names, types, ranges, semantic meanings, units, and available categorical values and their meanings. The prompt requests relational constraints involving feature i, excluding simple type, non-null, and declared-range checks. Each response contains a short description and a Python function returning whether a row satisfies the constraint. An atomic constraint describes one coherent relationship; unrelated checks should be returned separately.

Model and independent contexts. Generation uses `o3-2025-04-16`. Its documented knowledge cutoff is June 1, 2024, preceding the TabularBench preprint of August 14, 2024.[4] This motivates choosing o3 to limit exposure to the published golden constraints, which are never supplied during mining. Each feature–pass pair starts a new conversation without constraints from other pairs.

Validation, repair, and refinement. Prompt examples, repair examples, and clean acceptance rows have separate roles; final attack-evaluation rows are not used. A candidate must execute without errors and satisfy more than 99.9% of the acceptance rows. Upon failure, one repair prompt supplies the description, code, an execution error when present, and passing/failing examples from the repair split. It requests minimal changes preserving the intended relationship; refusal, unusable output, or failure after revalidation discards the proposal. Refinement receives the accepted constraints from the current conversation and requests an improvement or an additional relationship. Every proposal follows the same acceptance procedure; an unsuccessful revision leaves the earlier accepted constraint unchanged.

Semantic judgment and deduplication. The same LLM model is prompted as a domain professional to assess whether each accepted constraint is semantically reasonable, returning `True`, `False`, or `Uncertain`. The reported sets retain only initial `True` judgments. Description embeddings retrieve up to five similar constraints per candidate at cosine similarity at least 0.3; an LLM proposes

[4] `https://developers.openai.com/api/docs/models/o3`; `https://arxiv.org/abs/2408.07579`.

implication directions, which Z3 (de Moura & Bjørner, 2008) verifies within schema bounds. Mutual implications form equivalence groups. We retain one representative of each strongest group and remove constraints implied by those retained.

C.1 Complete algorithm for constraint generation

Algorithm 1 Constraint generation and verification

Require: Feature descriptions $S = \{s_i\}_{i=1}^{d}$; prompt examples $\mathcal{D}_{\text{prompt}}$; repair rows $\mathcal{D}_{\text{repair}}$; acceptance rows $\mathcal{D}_{\text{hold}}$
Ensure: Learned constraints $\mathcal{C}_{\text{LLM}}$ and equivalence-test outcomes E

$\mathcal{C} \leftarrow \varnothing$; $(R_1, R_2) \leftarrow (1, 6)$
for $p = 1, 2$ **do**
  for $i = 1, \ldots, d$ **do**
    Start an independent conversation
    $c \leftarrow \textsc{Propose}(i, S, \mathcal{D}_{\text{prompt}})$
    $V \leftarrow \textsc{ValidateRepair}(c, i, \mathcal{D}_{\text{repair}}, \mathcal{D}_{\text{hold}})$
    for $r = 1, \ldots, R_p$ **do**
      $c \leftarrow \textsc{Refine}(i, S, \mathcal{D}_{\text{prompt}}, V)$
      $V \leftarrow \textsc{UpdateAccepted}(V, c, i, \mathcal{D}_{\text{repair}}, \mathcal{D}_{\text{hold}})$
    end for
    $\mathcal{C} \leftarrow \mathcal{C} \cup V$
  end for
end for
$\mathcal{C} \leftarrow \textsc{JudgeFilter}(\mathcal{C})$
$\mathcal{C}_{\text{LLM}} \leftarrow \textsc{Deduplicate}(\mathcal{C})$
$E \leftarrow \textsc{EquivalenceTest}(\mathcal{C}_{\text{LLM}})$
return $(\mathcal{C}_{\text{LLM}}, E)$

C.2 Constraint-Specific Invalid Data Generation

For each learned validator c, we construct $\mathcal{D}^{c}_{\text{invalid}}$ to test whether a translation rejects rows that the original validator rejects. These probes target learned constraints and do not use the hidden golden constraints. Translation checks use a deterministic prefix of the first 100 anchors for invalid-data generation.

Perturbation searches. For an anchor satisfying c, we vary features referenced by c within their declared ranges, respecting integer and categorical encodings. Single-feature searches scan toward lower and upper bounds, refine violating values by bisection, and sample random replacements. We supplement these searches with boundary probes derived from features and constants in the validator, including coordinated feature changes to exercise conditional relationships. The configuration uses 50 directional steps, up to 40 bisection iterations, 256 boundary probes per constraint.

Retention and verification. We retain a probe only if c returns false both on the materialized row and after conversion to the float32 representation used by the translated evaluator; execution errors do not count as violations. We retain at most 100 invalid rows per constraint and use the same rows to check the solver and evaluation translations. A probe may violate other learned constraints as well: only violation of its target c is required.

If no invalid row is found, we attempt exhaustive enumeration for supported bounded integer domains of at most 10^6 assignments. Proven schema tautologies receive corresponding always-satisfied translations, otherwise, accepting a translation without negative probes requires a successful bounded symbolic equivalence check, and an inconclusive check rejects it. Finite clean/invalid-row agreement alone provides empirical evidence of agreement, not a proof over the full domain.

C.3 Complete Constraint-Projection and Recovery Algorithm

Algorithms 2–4 integrate the base attack, cyclic projection, local LLM assistance, recovery, and constraint-informed restarts. The original example x remains the reference for every perturbation-budget check. Let $V(v) = \{c \in \mathcal{C}_{\text{LLM}} : c(v; x) = 0\}$. The predicate FinalAccept requires

structural validity, $V(v) = \varnothing$, $D(v, x) \leq \epsilon$, and $f(v) \neq y$. All constraint checks use the original Python predicates.

Algorithm 2 Attack, cyclic projection, and constraint-informed restarts

Require: Original example (x, y); classifier f; base attack B; constraints; budget ϵ; restart limit R; cyclic limits P_m
Ensure: Accepted adversarial example or failure $\perp$

$a \leftarrow x$; $F \leftarrow \varnothing$ ▷ Restart anchor and base-attack freezes
for $k = 0, \ldots, R$ **do**
  $v \leftarrow B(a, F; x, \epsilon)$ ▷ Structural constraints and original budget
  If $v = \perp$, $f(v) = y$, or $D(v, x) > \epsilon$, return $\perp$
  Initialize repair history and exhausted-recovery records
  for $m \in (\text{ordinary}, \text{LLM}, \text{LLM+recovery})$ **do**
    $g \leftarrow (m \neq \text{ordinary})$; $h \leftarrow (m = \text{LLM+recovery})$
    for $\ell = 1, \ldots, P_m$ **do**
      If FINALACCEPT(v), return v
      If $V(v) = \varnothing$, end this stage
      $n \leftarrow |V(v)|$
      for each currently violated constraint c **do**
        $(p, \mathcal{A}, S) \leftarrow$ LOCALREPAIR$(v, \{c\}, g)$
        if $p \neq \perp$ and $f(p) = y$ **then**
          $p \leftarrow$ RECOVER$(v, \mathcal{A}, S)$ if h, otherwise $\perp$
        end if
        if $p \neq \perp$ and $|V(p)| < |V(v)|$ **then**
          Record the repair; $v \leftarrow p$; end this constraint pass
        end if
      end for
      If $|V(v)| \geq n$, end this stage
    end for
  end for
  If FINALACCEPT(v), return v
  If $k = R$, return $\perp$
  $c^\star \leftarrow$ exhausted constraint with greatest accumulated positive distance increase
  $i^\star \leftarrow$ eligible feature of $c^\star$ with greatest distance contribution
  If no such feature exists, return $\perp$
  $a \leftarrow$ retained repair point; if $D(a, x) > \epsilon$, set $a \leftarrow x$
  Add feature $i^\star$ to F, retaining earlier freezes at their restart values
end for
return $\perp$

Stopping criteria. A cyclic stage stops when no learned constraint remains violated, a complete pass produces no decrease in $|V(v)|$, or its pass limit is reached. The stages enable ordinary repair, then LLM-assisted repair, then LLM assistance with misclassification-aware recovery. Where enabled, full-group Couenne repair is attempted after local stagnation and must pass the same final acceptance checks. Recovery stops on success, its iteration limit, insufficient recent improvement in the best repair distance, or exhaustion of the classifier-query budget. The outer procedure stops on final acceptance, base-attack failure, its restart limit, or the absence of an eligible feature to freeze. Query-budget exhaustion terminates the entire procedure. Solver calls also have a time limit. The limits P_m, H, and R denote stage-specific cyclic passes, recovery iterations, and additional base-attack attempts, respectively.

Algorithm 3 Growing local constraint groups and LLM-assisted repair

function LOCALREPAIR($v, \mathcal{A}, g$)
  $\mathcal{M} \leftarrow \mathcal{A}$; $\mathcal{I} \leftarrow \varnothing$; $V_0 \leftarrow V(v)$ ▷ Main and side constraints
  $S \leftarrow$ mutable numerical features referenced by $\mathcal{M}$
  loop
    $\mathcal{A} \leftarrow \mathcal{M} \cup \mathcal{I}$
    $p \leftarrow$ SOLVEGROUP($v, \mathcal{A}, S, g$)
    if $p = \perp$ **then**
      if blocking side constraints $\mathcal{B} \subseteq \mathcal{I}$ can be identified **then**
        $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{B}$; $\mathcal{I} \leftarrow \mathcal{I} \setminus \mathcal{B}$
        Expand S to the mutable numerical features of $\mathcal{M}$; retry
      else
        return $(\perp, \mathcal{A}, S)$
      end if
    else
      $E \leftarrow V(p) \setminus V_0$ ▷ Recheck all learned constraints
      If $E = \varnothing$, return $(p, \mathcal{A}, S)$
      $\mathcal{I} \leftarrow \mathcal{I} \cup E$ ▷ Add newly broken constraints; keep S fixed
    end if
  end loop
end function
function SOLVEGROUP($v, \mathcal{A}, S, g$)
  $T \leftarrow$ current solver representations for $\mathcal{A}$
  $p \leftarrow$ COUENNE(v, x, T, S) ▷ Two-term local repair objective
  If $p \neq \perp$ and LOCALVALID($p, v, \mathcal{A}, S$), return p
  If $g =$ false, return $\perp$
  for $j = 1, 2, 3$ **do**
    $T' \leftarrow$ LLMREFORMULATE($T, \mathcal{A}, S$, failure feedback)
    $p \leftarrow$ COUENNE(v, x, T', S)
    if $p \neq \perp$ and LOCALVALID($p, v, \mathcal{A}, S$) **then**
      Retain the validated reformulation; **return** p
    end if
    Discard the unsuccessful reformulation and update failure feedback
  end for
  $p \leftarrow$ LLMVALUESONCE($v, \mathcal{A}, S$, minimal change)
  If $p \neq \perp$ and LOCALVALID($p, v, \mathcal{A}, S$), return p
  return $\perp$
end function

Growing the local group. Each local repair starts with one violated constraint in the main set $\mathcal{M}$ and an empty set $\mathcal{I}$ of immutable constraints. After each successful local solve, we recheck all learned constraints and add those satisfied at the local input v but violated at the proposed repair p to $\mathcal{I}$. These constraints are added to the solver and enforced together with $\mathcal{M}$, but S is unchanged: their features outside S remain fixed at their values in v. We retry this joint repair before releasing any additional features. Only if it fails do we use validation feedback or promotion probes to select blocking immutable constraints $\mathcal{B}$ for promotion. We move $\mathcal{B}$ from $\mathcal{I}$ into $\mathcal{M}$, expand S to release their additional mutable numerical features, and retry from the same local input v. Rechecking can reveal further newly broken constraints, which are added to $\mathcal{I}$ even if the total number of violations stays the same or decreases; other constraints already violated at v are handled by the outer cyclic pass.

Local repair and acceptance. LOCALVALID checks structural constraints, the current group $\mathcal{A}$, and that features outside S remain unchanged. Categorical and immutable features are fixed; under our encoding and $\epsilon = 0.5$, categorical changes already exceed the perturbation budget. Local repair uses the two-term objective in Section 4.2. When ordinary repair fails, we allow up to three LLM-assisted reformulation attempts. If all three fail, we query the LLM once for a satisfying assignment with minimal range-normalized change over S. Returned rows must pass LOCALVALID; unsuccessful reformulations are discarded. These row checks do not constitute a global equivalence proof. Local acceptance permits continued projection; only FINALACCEPT permits returning an attack.

Algorithm 4 Label-only recovery after a repair loses misclassification

Require: Samples per iteration N; iteration limit H; initial spread r_0;
maximum spread $r_{\max}$; stagnation window and improvement threshold

function RECOVER($v, \mathcal{A}, S$)
  If $S = \varnothing$, record exhausted recovery and return $\perp$
  $a \leftarrow v$; $r \leftarrow r_0$
  for $t = 1, \ldots, H$ **do**
    $\mathcal{P} \leftarrow \varnothing$
    for $j = 1, \ldots, N$ **do**
      $q \leftarrow$ Gaussian perturbation of a on S, scaled by r and feature ranges
      Enforce numerical bounds and integer types; keep features outside S fixed
      If $f(q) = y$, continue to the next sample
      If LOCALVALID($q, v, \mathcal{A}, S$) and $D(q, x) \leq \epsilon$, return q
      $p \leftarrow$ SOLVEGROUP($q, \mathcal{A}, S,$ true)
      If $p = \perp$, continue to the next sample
      If $f(p) \neq y$ and $D(p, x) \leq \epsilon$, return p
      $\mathcal{P} \leftarrow \mathcal{P} \cup \{(q, p)\}$
    end for
    if $\mathcal{P} \neq \varnothing$ **then**
      $(q^\star, p^\star) \leftarrow \arg\min_{(q,p)\in\mathcal{P}} \|W(q - p)\|_2$
      $a \leftarrow q^\star$; $r \leftarrow \min(2r, r_{\max})$
    end if
    Update recent score improvement; if stagnated, end recovery
  end for
  Record exhausted recovery for the targeted constraints and their mutable features
  return $\perp$
end function

Recovery and restart details. For $s = |S| > 0$, recovery draws $z \sim \mathcal{N}(0, I_s)$ and perturbs the search center by $rW_S^{-1}z/\sqrt{s}$ on S, then clips numerical bounds and rounds integer values. An unsuccessful iteration with valid repair pairs recenters on the adversarial sample q minimizing $\|W(q-p)\|_2$ and increases the sampling spread. To select a restart constraint, we accumulate positive increases in $D(\cdot, x)$ over its recorded repairs and rank constraints whose recovery was exhausted. Among its eligible features, we select the one whose hypothetical reset to its original value most reduces distance. This reset is used only for ranking. The feature is frozen at its restart value during subsequent base-attack steps; earlier freezes are retained. Projection can still modify these attack-frozen features. The restart anchor is the retained repaired point, with a clean-anchor fallback if it exceeds the original budget.

C.4 WORKED RUNNING EXAMPLE

The boxed running example in §4.1 and §4.2 is a constructed illustration, not a benchmark record or a measured LLM/solver execution. Its samples illustrate possible search outcomes; they are not draws reported from a particular random seed. The local optima below follow the stated two-term objective and can be checked by finite enumeration.

Schema, classifier, and acceptance. Write each row in the order (O, T, C, r, V), where O, T, C count open, total, and closed accounts, r is the open-account ratio, and V counts verified accounts. All five features are mutable; any omitted features remain fixed. The structural domains are

$$O, T, C \in \{0, \ldots, 20\}, \qquad V \in \{0, \ldots, 10\}, \qquad r \in [0, 1].$$

In this hypothetical account system, every account is either open or closed, and every open account must be verified. The learned predicates are

$$c_R(z): \quad \begin{cases} O = 0 \ \wedge \ r = 0, & T = 0, \\ r = O/T, & T > 0, \end{cases}$$
$$c_P(z): \quad T = O + C, \qquad c_V(z): \quad O \leq V.$$

Equalities are exact in this illustration. The original row, metric, and perturbation budget are

$$x_{\texttt{origin}} = (2, 10, 8, 0.2, 2), \qquad W = \mathrm{diag}(1/20, 1/20, 1/20, 1, 1/10),$$
$$D(z, x_{\texttt{origin}}) = \|W(z - x_{\texttt{origin}})\|_2, \qquad \epsilon = 0.5.$$

Let $f(z) = \mathbf{1}[2O - V \geq 5]$ and $y = f(x_{\texttt{origin}}) = 0$. Thus a row is adversarial precisely when $2O - V \geq 5$. Final acceptance additionally requires all three learned predicates, structural validity, and the original perturbation budget.

Generation, repair, refinement, and deduplication. One possible generation history starts with the strict rule $O < T$. A valid clean row $(5, 5, 0, 1, 5)$ contradicts it, prompting the single allowed repair to $O \leq T$, followed by revalidation. Refinement can add the more informative guarded ratio rule c_R; independent feature queries supply c_P and c_V. Each proposal undergoes the ordinary validation and semantic checks. Once the stronger predicates are retained, $O \leq T$ is redundant: c_P implies it because $C \geq 0$, and c_R also implies it because $r \leq 1$, including the zero-account case. These implications hold over the declared domains, rather than merely on the displayed clean rows. The final example has three retained relationships; removing the weaker bound does not merge these distinct relationships.

Translation and the conditional LLM-assistance branch. A Python validator can branch on $T = 0$ before evaluating O/T. If a guarded quotient formulation causes a solver failure at the zero-denominator boundary, an LLM could instead propose

$$\widetilde{c}_R(z): \quad O = rT \ \wedge \ r \leq T.$$

This is equivalent to c_R under the example's structural domains. At $T = 0$, the product equality forces $O = 0$, while $0 \leq r \leq T$ forces $r = 0$. For positive integer T, $T \geq 1$, so $r \leq T$ already follows from $r \leq 1$, and the product equality is equivalent to the quotient relation. Dropping $r \leq T$ would change the rule: the structurally valid probe $(0, 0, 0, 1, 0)$ satisfies $O = rT$ but is rejected by c_R. A second invalid probe, $(2, 10, 8, 0.3, 2)$, exercises the positive-denominator branch. Clean/invalid-row agreement tests the executable translation; the case analysis above separately establishes equivalence for this toy rule. An actual validator's tolerance and zero-case convention must also be preserved. This branch illustrates a possible reformulation, not an observed numerical failure. If translation already produces this form, the online reformulation is unnecessary. If reformulation attempts fail, an LLM-proposed row such as p_0 below must still pass the original active predicates and structural checks; satisfying those checks alone does not establish attack success.

Initial cyclic repair. For any current candidate a, let

$$J(z; a) = \tfrac{1}{2}\|W(z - a)\|_2^2 + \tfrac{1}{2}\|W(z - x_{\texttt{origin}})\|_2^2.$$

The base attack produces $v = (6, 10, 4, 0.2, 3)$, with $D(v, x_{\texttt{origin}}) = 0.300$. It is adversarial and satisfies c_P, but violates c_R, c_V. Target c_R first with $S = \{O, T, r\}$, keeping $C = 4, V = 3$. Minimizing $J(\cdot; v)$ under c_R gives $(2, 10, 4, 0.2, 3)$, which breaks the previously satisfied c_P. Add c_P to $\mathcal{A}$. Its additional feature C can remain fixed: the joint problem is feasible without releasing it. The joint optimum is

$$p_0 = (2, 6, 4, 1/3, 3), \qquad D(p_0, x_{\texttt{origin}}) \simeq 0.328.$$

Its classifier score is $2O - V = 1$, so recovery is needed. The verification rule c_V was already violated at v; it was not a previously satisfied relation broken by this local repair, and hence has not entered the active group $\{c_R, c_P\}$.

Failed pair, reanchoring, and successful local recovery. Suppose the best projectable adversarial sample in one recovery iteration is

$$q_1 = (5, 9, 4, 0.4, 3), \qquad p_1 = (3, 7, 4, 3/7, 3).$$

The active group remains $\{c_R, c_P\}$, and only O, T, r may change. Every feasible local repair has the form

$$z(o) = \big(o, o + 4, 4, o/(o + 4), 3\big), \qquad o \in \{0, \ldots, 16\}.$$

Enumerating $J(z(o); a)$ gives unique minimizers $o = 2$ for $a = v$, $o = 3$ for $a = q_1$, and $o = 4$ for $a = q_2$ defined below. Although q_1 is adversarial, p_1 is not: their classifier scores are 7 and 3. Nevertheless, the repair gap falls from $\|W(v - p_0)\|_2 \simeq 0.313$ to $\|W(q_1 - p_1)\|_2 \simeq 0.144$. This gap measures the movement needed to repair an adversarial sample, not its distance from the original row.

Recovery reanchors at q_1, rather than p_1, and increases its sampling spread subject to the configured cap. A subsequent sample and its repair are

$$q_2 = (5, 9, 4, 0.6, 3), \qquad p_2 = (4, 8, 4, 0.5, 3).$$

The gap is now approximately 0.122. Both rows are adversarial; p_2 satisfies the active relations and $D(p_2, x_{\texttt{origin}}) = 0.400$. The sampled rows themselves also lie within budget: $D(q_1, x_{\texttt{origin}}) \simeq 0.339$ and $D(q_2, x_{\texttt{origin}}) \simeq 0.485$. Recovery therefore returns p_2 to the cyclic procedure. The number of violated learned predicates decreases from two at v to one at p_2, but c_V remains violated, preventing final acceptance.

The next constraint and exhausted recovery. Target c_V at p_2 with $S = \{O, V\}$. Its first local optimum, $(3, 8, 4, 0.5, 3)$, breaks both c_R and c_P, so they join the active group. With $T = 8, C = 4, r = 0.5$ still fixed, these relations require $O = 4$. The joint optimum is

$$p_3 = (4, 8, 4, 0.5, 4), \qquad D(p_3, x_{\texttt{origin}}) \simeq 0.436.$$

Its classifier score is 4, so misclassification is lost. Every feasible repair in this local problem has $O = 4, V \geq 4$, implying $2O - V \leq 4$. Consequently, recovery over $S = \{O, V\}$ cannot succeed and terminates at its local iteration or stagnation limit. Classifier queries remain available for a restart. This is a limitation of the current local problem, not global infeasibility. If a full-group minimum-distance fallback is attempted at p_2, its optimum $(3, 9, 6, 1/3, 3)$ is also non-adversarial.

Blame, the base-attack freeze, and final acceptance. The c_V-targeted repair increased distance from 0.400 to approximately 0.436. It is the only target with exhausted recovery in this trace, so it is selected for blame. At the retained point p_3, the eligible features' normalized displacements from the original are 0.1 for O and 0.2 for V. Hypothetically resetting them reduces the full distance by approximately 0.0116 and 0.0486, respectively. Thus V is frozen at its restart value 4 during the next base attack; the hypothetical reset is used only for ranking. Starting from p_3, a possible base-attack result is

$$v^+ = (5, 10, 5, 0.5, 4), \qquad D(v^+, x_{\texttt{origin}}) \simeq 0.418.$$

It respects the freeze, is adversarial, and satisfies c_R, c_P, but violates c_V. The first repair of c_V, $(3, 10, 5, 0.5, 3)$, again breaks the other two relations. Adding them while keeping $T = 10, C = 5, r = 0.5$ fixed forces $O = 5$, and joint repair yields

$$p^+ = (5, 10, 5, 0.5, 5), \qquad D(p^+, x_{\texttt{origin}}) \simeq 0.474 < 0.5.$$

All learned predicates now hold, and $2O - V = 5$, so the attack succeeds. Projection is allowed to change V despite the base-attack freeze. Every distance and budget check remains anchored to $x_{\texttt{origin}}$.

D Label-Only Attack Extensions

D.1 Label-Only CGBA Full Algorithm: With Extensions

Algorithm 5 combines multi-direction search, the boundary walk, and the CGBA step through the following operations:

- InitializeAdversarial: finds a starting point misclassified by the model, or returns failure ($\perp$).
- BoundaryRefine: uses binary search between the clean input and an adversarial point to approach the decision boundary, returning the last adversarial endpoint.
- EstimateNormal: probes around the current boundary point and combines the perturbation directions, signed by whether their labels are adversarial, to estimate a boundary-normal direction.
- SearchDirections (Appendix D.4): selects normal-based, coordinate-based, and random directions at the first search location; subsequent locations use fresh random directions.
- CGBA step (Appendix D.2): searches a semicircle in one selected plane and returns the closest adversarial candidate it queried, or $\perp$ if none was found.
- BoundaryHop (Appendix D.3): moves outward in a new direction until an adversarial probe is found, then calls BoundaryRefine to obtain the next exploration point. A failed hop returns $\perp$.

Every candidate is structurally projected before querying the model. The returned base-attack candidate undergoes the learned-constraint checks and recovery described in Appendix C.3.

Algorithm 5 Label-only CGBA with multiple directions and boundary hops

Require: Clean input (x, y); label oracle f; normalized distance D;
structural projector; query budget Q;
outer iteration limit I; maximum search locations H
Ensure: Best retained adversarial candidate b, or failure

$b \leftarrow \text{INITIALIZEADVERSARIAL}(x, y)$
if $b = \perp$ **then**
  return failure
end if
$b \leftarrow \text{BOUNDARYREFINE}(x, b)$
for $t = 1, \ldots, I$ while queries remain **do**
  $\widehat{n} \leftarrow \text{ESTIMATENORMAL}(x, b, y)$
  $u \leftarrow b$; $b^\star \leftarrow b$
  for $h = 0, \ldots, H-1$ while queries remain **do**
    $\mathcal{D}_h \leftarrow \text{SEARCHDIRECTIONS}(h, u, \widehat{n})$
    for each $d \in \mathcal{D}_h$ while queries remain **do**
      $c \leftarrow \text{CGBA STEP}(x, u, d)$
      if $c \neq \perp$ and $D(c, x) < D(b^\star, x)$ **then**
        $b^\star \leftarrow c$
      end if
    end for
    if $h < H - 1$ and queries remain **then**
      $u' \leftarrow \text{BOUNDARYHOP}(x, u)$
      if $u' = \perp$ **then**
        break
      end if
      $u \leftarrow u'$
    end if
  end for
  $c \leftarrow \text{BOUNDARYREFINE}(x, b^\star)$
  if $f(c) \neq y$ and $D(c, x) < D(b, x)$ **then**
    $b \leftarrow c$
  end if
end for
return b

Here u is the exploration point, $b^\star$ the best candidate within the current walk, and b the retained outer candidate. All label queries share Q. Per-query budget checks are omitted from the pseudocode for brevity; execution stops when Q is exhausted and returns the last certified retained candidate.

D.2 CGBA STEP

We reformulate the CGBA step of Reza et al. (2023) using our notation and structural constraints consideration. The goal is to find an adversarial point closer to the clean input x, starting from an adversarial point u. Let W scale features into normalized coordinates, and define $r = \|W(u - x)\|_2$ (current distance) and $v = W(u - x)/r$ (unit direction from x to u).

Constructing the search path. Let d be a unit search direction in normalized coordinates. Original CGBA uses an estimated boundary-normal direction; our additional directions are described in Appendix D.4. Remove the component of d parallel to v and normalize:

$$\tau = \frac{d - (d^\top v)v}{\|d - (d^\top v)v\|_2}.$$

Thus τ points sideways relative to v, and the two perpendicular directions define the search plane. If the denominator is zero, this direction cannot define a plane and is skipped.

For an angle ϕ measured from v, propose the normalized displacement

$$z(\phi) = r \cos\phi \left(\cos\phi\, v + \sin\phi\, \tau\right), \qquad 0 \leq \phi < \frac{\pi}{2}.$$

The expression in parentheses rotates the direction by ϕ; the factor $r\cos\phi$ simultaneously shortens the displacement. In particular,

$$\|z(\phi)\|_2 = r\cos\phi, \qquad \left\|z(\phi) - \frac{r}{2}v\right\|_2 = \frac{r}{2}.$$

The path therefore follows a semicircle with diameter from the clean input to u in normalized coordinates. It starts at u when $\phi = 0$ and approaches x as ϕ approaches $\pi/2$.

Finding an adversarial point on the path. Each proposal is mapped back to feature coordinates and structurally projected:

$$p(\phi) = P_{\text{struct}}\big(x + W^{-1}z(\phi); x\big).$$

The search probes increasing angles and tests whether $f(p(\phi)) \neq y$. It retains the closest adversarial candidate encountered. Once a probe is non-adversarial, its angle and the initial adversarial angle $\phi = 0$ form a bracket. Angular bisection tests the midpoint angle and replaces the endpoint with the same adversarial status, repeatedly narrowing the bracket. Every adversarial midpoint is also considered for the retained candidate.

The step returns the closest adversarial candidate it queried, or failure if none was found. Structural projection can change the ideal geometry, so both distance and adversariality are checked on the projected candidate. The procedure seeks a local improvement; it does not assume that every shorter proposal remains adversarial.

D.3 Multi-Hop Boundary Walk

A local boundary search can stall when reaching a better region requires first moving farther from the clean input. The multi-hop boundary walk permits these temporary increases in distance while retaining the best adversarial candidate found by its local searches. Each hop chooses a new direction, finds an adversarial point farther along it, and contracts toward the clean input to locate a new boundary point.

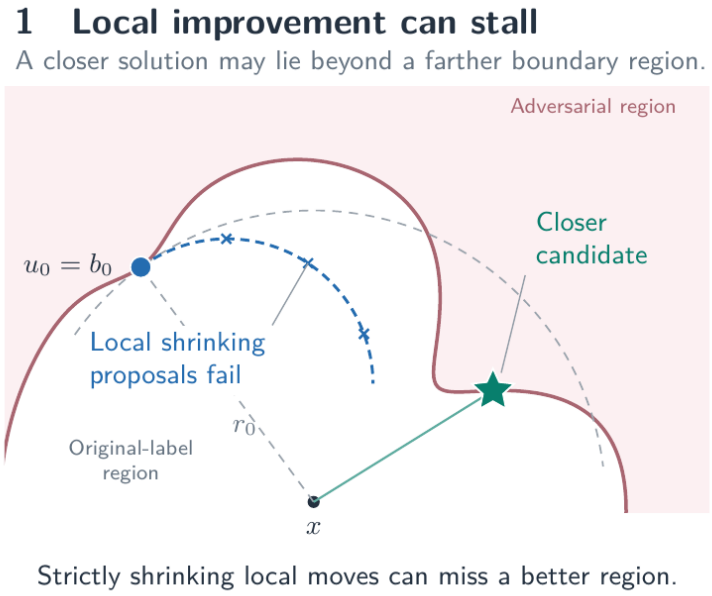

Figure 1: Multi-Hop - Stall

Exploration point and retained candidate. Let x (clean input) satisfy $f(x) = y$, where f (label-only classifier) returns y (the correct label). At hop h (walk index), let u_h (current exploration point) be adversarial and let b_h (best retained candidate) initially satisfy $u_0 = b_0$. With W (diagonal matrix

of inverse feature scales), define

$$D(u,x) = \|W(u-x)\|_2, \qquad r_h = D(u_h, x), \qquad v_h = \frac{W(u_h - x)}{r_h}.$$

Here D (normalized distance) compares perturbations across feature units, r_h (exploration radius) measures the distance from x to u_h, and v_h (unit radial direction) points from x toward u_h in normalized coordinates. Every distance remains measured from the original x.

Rotate and move outward. Draw t_h (random unit lateral direction) perpendicular to v_h within the available feature subspace, aiming to explore a new direction around x rather than move directly toward or away from it. Using $\alpha > 0$ (rotation magnitude), construct w_h (new unit exploration direction) and $q_h(\delta)$ (outward probe):

$$w_h = \frac{v_h + \alpha t_h}{\sqrt{1+\alpha^2}}, \qquad q_h(\delta) = P_{\text{struct}}\left(x + (1+\delta) r_h W^{-1} w_h; x\right), \quad \delta > 0.$$

Here δ (relative outward expansion) increases the proposed radius, W^{-1} (inverse scaling) restores original feature units, and P_{struct} (structural projection) enforces feature constraints. The rotation angle is $\arctan \alpha$. Because t_h is tangent to a sphere around x, not necessarily to the decision boundary, rotation alone can leave the adversarial region. Outward expansion seeks an adversarial endpoint in the new direction. We try increasing displacements and retain the first q_h with $f(q_h) \neq y$. If none succeeds, the walk stops.

Return toward the boundary and search again. Binary search between x and q_h repeatedly tests a structurally projected midpoint. An adversarial midpoint replaces the adversarial endpoint; otherwise it replaces the non-adversarial endpoint. The last adversarial endpoint becomes

$$u_{h+1} = \text{BoundaryRefine}(x, q_h),$$

where BoundaryRefine (label-based boundary refinement) removes excess outward displacement. Integer rounding can limit precision, and multiple label transitions need not yield the closest crossing. Crucially, $D(u_{h+1}, x)$ may exceed $D(u_h, x)$. Local CGBA search then explores this new boundary location, which a strictly improving trajectory might never reach.

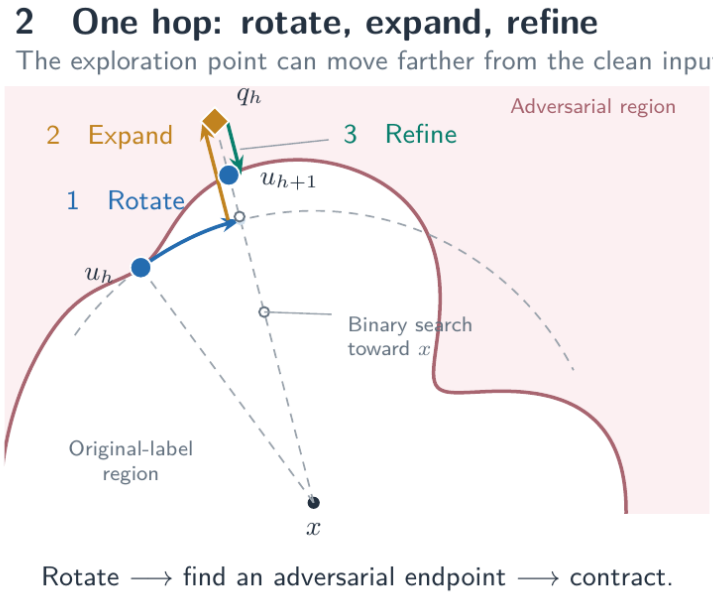

Figure 2: Multi-Hop - One Hop

Keep improvements without restricting the walk. Let $\mathcal{S}_h$ (adversarial candidates found by local searches at u_h) supply the retained-point update:

$$b_{h+1} \in \underset{u \in \{b_h\} \cup \mathcal{S}_h}{\arg\min} D(u, x), \qquad D(b_{h+1}, x) \le D(b_h, x).$$

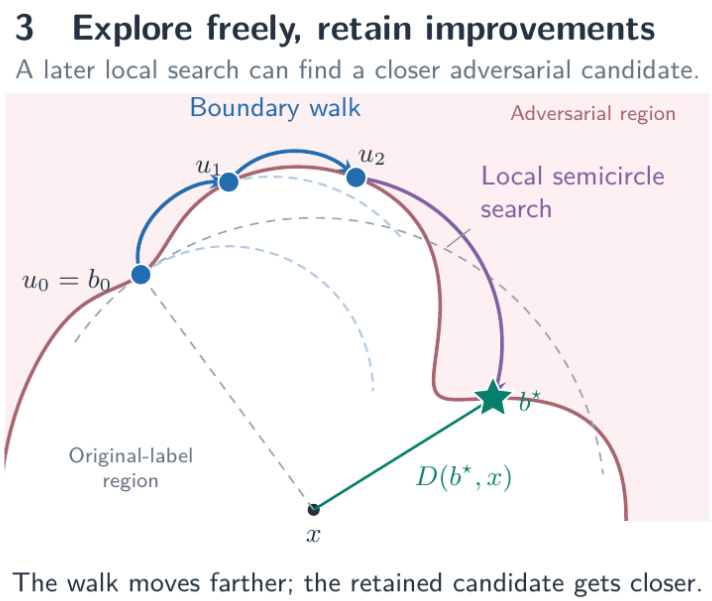

Figure 3: Multi-Hop - Many Hops

Thus exploration may worsen distance while the retained result never does. The next hop originates from u_h, independently of improvements to b_h. Repeated hops change the boundary locations explored without guaranteeing a global optimum. After the walk, the retained candidate is refined toward x and adopted only if it remains adversarial and improves the pre-walk result. Final acceptance also requires the perturbation budget and learned constraints.

D.4 Multi-Direction Semicircle Search.

We broaden CGBA's semicircle search (Reza et al., 2023) by considering coordinate and random directions alongside the estimated boundary normal. This reduces reliance on a single normal estimate and provides alternative search planes when local progress stalls.

All directions are constructed in scaled feature coordinates. At hop h, let $\widehat{v}_h$ denote the unit direction from the clean input to the current boundary point, and let M mask features that cannot be modified. For a proposed direction a, we construct

$$T_h(a) = \frac{Ma - (\widehat{v}_h^\top Ma)\widehat{v}_h}{\left\| Ma - (\widehat{v}_h^\top Ma)\widehat{v}_h \right\|_2}.$$

Directions with a zero or numerically negligible denominator are discarded. This construction makes $T_h(a)$ orthogonal to the radial direction $\widehat{v}_h$.

At the first hop, the search directions comprise the estimated unit normal $\widehat{n}$, coordinate directions $T_0(e_j)$, and random directions $T_0(g_k)$:

$$\mathcal{D}_0 = \{\widehat{n}\} \cup \{T_0(e_j) : j \in J\} \cup \{T_0(g_k) : k = 1, \dots, K\},$$

where e_j is a coordinate basis vector, J contains up to twelve modifiable coordinates sampled without replacement, and each g_k is an independent standard Gaussian vector. Subsequent hops use newly sampled random directions.

Each direction defines a search plane with $\widehat{v}_h$, within which we perform the CGBA semicircle search and hop label-based boundary-refinement steps when a bracket is found.