RELATIONAL INJECTION ATTACK AGAINST RELATIONAL DEEP LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Relational deep learning (RDL) predicts outcomes for entities using information from database rows connected through foreign-key relationships. This reliance on relational context raises a question: can permitted row insertions manipulate an existing target's prediction without modifying existing records or the trained model? We formulate Relational Injection Attack (RIA), a test-time attack that inserts linked rows under explicit write permissions. Under a white-box setting, our framework consists of two components: schema-admissible path selection and value optimization. The path selector ranks injection paths using the model's response and the potential gains from attribute changes. For value optimization, we introduce Relational Projected Gradient Descent (RelPGD), which optimizes continuous weights over discrete candidate values using centered and scaled gradients, followed by multi-start local search on unsuccessful targets. We evaluate twelve binary classification tasks from seven RelBench datasets. Our path selector improves macro-average attack success rate by 10.1%p over the relation-weight-norm heuristic. RelPGD matches or exceeds the mean attack success rates of CAPGD in 21 of 24 task–model settings. These findings demonstrate that permitted row injections can affect predictions, motivating their inclusion in RDL robustness evaluations. For transparency, our source code is available at: `https://anonymous.4open.science/r/RIA-DCC0`

1 INTRODUCTION

Relational databases (RDBs) are widely used to store and manage structured data across many domains. RDBs organize information across multiple tables connected by primary–foreign key relationships. Predicting an outcome for one entity therefore often requires evidence from rows in other tables: a product's future activity, for example, may depend on its reviews and their authors. Conventional machine learning pipelines incorporate such evidence by joining and aggregating rows into a feature vector for each prediction target. Relational Deep Learning (RDL) instead learns representations directly from the database structure, treating rows as nodes and primary–foreign key links as edges (Fey et al., 2024; Robinson et al., 2024). In this relational entity graph (REG), each row's source table, attributes, timestamp, and foreign key values determine the corresponding node's type, features, time, and typed edges. Message passing allows information from related entities to contribute directly to a target's prediction.

However, GNN-based models are widely known to be vulnerable to even small adversarial perturbations to the input graph (Zügner et al., 2018; Dai et al., 2018). This raises a corresponding question: can an attacker manipulate a prediction by inserting new rows around a target they cannot modify? In database-backed applications, attacker capabilities are governed by permissions over row creation and modification. Under creation-only permissions, the attacker cannot directly alter the target entity or its existing relationships, but can insert permitted rows that become part of its predictive context. The resulting REG is therefore modified only indirectly through database insertions, motivating row insertion as an attack surface for RDL.

Consider a platform that predicts product inactivity from reviews and their authors (Figure 1). An attacker may register a new customer and submit a review of the target product, requiring coordinated customer and review insertions to satisfy foreign-key constraints. We treat such jointly required rows as an *injection bundle*, the atomic attack action.

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.

Adding the bundle creates a new information path to the target without modifying the product or any existing rows. We define this setting as a *Relational Injection Attack (RIA)*, where new database rows satisfying explicit insertion constraints are added at test time while existing rows and label-defining observations remain unchanged. Because attacker capabilities vary across applications, the threat model specifies which rows may be inserted and which existing entities they may reference.

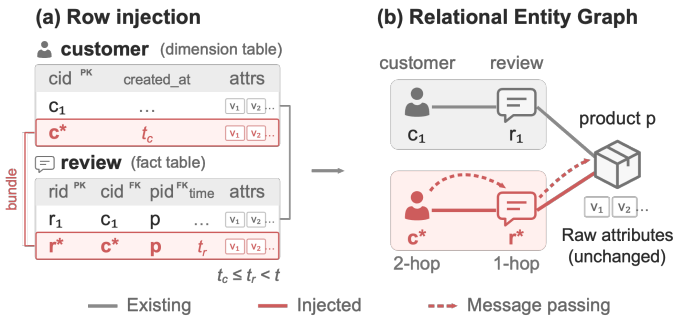

Figure 1: Example of RIA on item-churn prediction.

Unlike graph injection formulated directly over nodes and edges, relational injection must realize every graph change through permitted database writes. This introduces three requirements: a database-derived graph representation, schema-constrained structural placement, and valid values for the inserted rows. We address these requirements through two separate attack tasks.

(i) **Path selection** identifies schema-admissible primary–foreign key paths through which inserted rows can reach the target, thereby determining the required tables. Unlike existing graph injection attacks that directly optimize added nodes, edges, and features in graph space (Zou et al., 2021; Wang et al., 2022), RIA restricts these choices to relational injections realizable under schema and integrity constraints.

(ii) **Value optimization** assigns row values and timestamps satisfying data-type, domain, and temporal constraints. Existing constrained tabular attacks such as CAA (Simonetto et al., 2022; 2024) are not directly applicable because they perturb an existing input within a bounded neighborhood, whereas row insertion creates new rows directly from permitted domains without a reference row. We therefore introduce Relational Projected Gradient Descent (RelPGD) to optimize inserted values, followed by exploratory search for targets unresolved by the gradient stage.

The main contributions are summarized as follows:

- We formulate *Relational Injection Attack*, which manipulates predictions through new database rows under explicit insertion constraints and budgets, while preserving existing rows and label-defining observations.
- We develop an attack framework that separates path selection from value optimization, introducing *score-based path selection* and *RelPGD* to optimize the values of injected rows.
- We evaluate attack effectiveness and computational cost across relational prediction tasks and model architectures, comparing our methods with adapted baselines under shared insertion constraints and budgets.

2 Related Work

Adversarial robustness in relational deep learning. RDL represents database rows and their primary–foreign key relationships as a heterogeneous graph (Fey et al., 2024). These relational pathways create an attack surface: prior white-box attacks rewire existing foreign-key references while preserving integrity constraints (Gany et al., 2026). In contrast, we permit only valid row insertions and study whether they can flip a frozen predictor's decision for an existing target.

Graph injection attacks. Graph injection attacks manipulate predictions by adding nodes and edges to an existing graph. TDGIA studies topology-guided injection (Zou et al., 2021), while Fake Node Attacks demonstrates attacks through maliciously added nodes (Wang et al., 2022). However, graph-level injections do not by themselves guarantee realizability as database writes: a proposed node and its incident edges may not correspond to a row with valid column values and foreign-key references under the database schema and write permissions. In our setting, each injected node

and its foreign-key edges must therefore be induced by permitted row insertions into the underlying database.

Constrained adversarial attacks on tabular data. Constrained tabular attacks account for restrictions on feature values and dependencies. CPGD incorporates constraints into gradient-based optimization, while MOEVA uses a multi-objective genetic algorithm to seek misclassification, small perturbations, and constraint satisfaction (Simonetto et al., 2022). CAPGD introduces adaptive gradient mechanisms, and CAA combines CAPGD with the genetic-algorithm-based search of MOEVA (Simonetto et al., 2024).

3 PRELIMINARY

3.1 RELATIONAL DEEP LEARNING

Relational database. A relational database $\mathcal{D}$ comprises tables $\mathcal{T} = \{T_1, \ldots, T_n\}$ and foreign-key relations $\mathcal{R} \subseteq \mathcal{T} \times \mathcal{T}$. Rows represent entities, events, or associations and contain numerical, categorical, textual, or temporal attributes. Primary keys identify rows, while each $(T_i, T_j) \in \mathcal{R}$ denotes a foreign-key reference from T_i to T_j. Fact tables typically record events or associations, and dimension tables describe the referenced entities.

RDB-derived heterogeneous graph. We represent the database as a relational entity graph (REG) $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \phi, \psi, \tau)$ (Fey et al., 2024). Each row becomes a node and each foreign-key reference a directed edge; ϕ and ψ assign their table and relation types. Non-key attributes provide node features, and reverse edges enable bidirectional message passing. For a target queried at time t, temporal sampling uses τ to construct a target-centric subgraph containing only rows available by t. A GNN then combines node attributes and relational context to produce the prediction.

3.2 GRAPH ADVERSARIAL ATTACKS

Graph injection attacks. Graph adversarial attacks manipulate an input graph to influence model predictions. Injection attacks add new nodes, their attributes, and incident edges without modifying the original graph:

$$\mathcal{G}' = (\mathcal{V} \cup \mathcal{V}_{\text{inj}}, \mathcal{E} \cup \mathcal{E}_{\text{inj}}, \mathbf{X} \cup \mathbf{X}_{\text{inj}}), \tag{1}$$

where the three injected components denote the new nodes, edges, and attributes, respectively.

Attack settings. Attacks differ in objective, attacker knowledge, and phase. Global attacks degrade overall performance, whereas target-specific attacks focus on nodes $\mathcal{V}_{\text{tar}} \subseteq \mathcal{V}$; they may seek any incorrect class or a particular attacker-specified class. White-box attackers know the victim architecture, parameters, and input graph, while gray- and black-box attackers have partial knowledge or rely on outputs and surrogate models. Poisoning attacks alter training data, whereas evasion attacks modify only the inference-time input while keeping the trained parameters θ fixed.

4 RELATIONAL INJECTION ATTACK

We define *Relational Injection Attack* as a specialized form of graph injection attack by restricting injected nodes and edges to those induced by valid relational injections into the underlying relational database. Let $i = (v_i, t_i)$ denote a prediction instance defined by a target entity v_i and a prediction time t_i.

Insertion constraints. **(C1) Domain constraints** specify the tables into which new rows may be inserted, the writable columns whose values may be assigned by the attacker, and the permitted data types and value domains of those columns; **(C2) Referential constraints** specify which existing or newly inserted rows may be referenced through foreign keys and require these assignments to preserve referential integrity; **(C3) Temporal constraints** specify the permitted event times and determine when inserted rows become available to the victim; and **(C4) Budget constraints** limit the number of injection bundles to B per target.

A row insertion is represented as $r = \left(T_r, \mathbf{x}_r, \mathbf{k}_r, t_r\right)$, where T_r is the table into which r is inserted, $\mathbf{x}_r$ contains its attribute values, $\mathbf{k}_r$ contains its foreign-key assignments, and $t_r \leq t_i$ is its event time. The attack is restricted to inserting new rows. As these feasibility constraints are schema-dependent, relational attacker knowledge includes table definitions and primary–foreign key relationships but is distinct from access to the underlying records and attribute values (Gany et al., 2026).

Schema paths and injection bundles. Let $\mathcal{P}_i$ denote the set of schema paths through which inserted rows can reach the target entity v_i within the model's L-hop receptive field. Each candidate path $p \in \mathcal{P}_i$ is represented as $p = \left(T_i, T_p^{\text{fact}}, T_p^{\text{dim}}\right)$, where T_i is the table containing v_i, T_p^{fact} is a fact table permitted for insertion, and T_p^{dim} is a dimension table referenced by T_p^{fact}. The fact table contains foreign keys referencing both the target table and the dimension table. Consequently, inserting rows along p creates a relational path from the target entity to a newly inserted dimension entity through a newly inserted fact row.

The path p is then instantiated by an injection bundle $b = \left(r^{\text{dim}}, r^{\text{fact}}\right)$, where r^{dim} is inserted into T_p^{dim} and r^{fact} is inserted into T_p^{fact}. A relational injection attack with budget B is represented by $\mathbf{b} = (b_1, \ldots, b_B)$. Applying these bundles to the clean database produces the attacked database $\mathcal{D}' = \mathcal{D} \oplus \mathbf{b}$. The corresponding target-specific subgraph is denoted by $\mathcal{G}_i' = \mathcal{G}_i\left(\mathcal{D}'\right)$, where $\mathcal{G}_i(\cdot)$ denotes the time-consistent L-hop subgraph construction for i. In the resulting REG, each injected fact node is adjacent to the target node, while its associated dimension node lies one additional hop away.

Let f_θ denote the frozen victim model and let y_i be the ground-truth label of the target instance. A relational injection attack is then expressed as

$$\mathbf{b}_i^* = \underset{\mathbf{b} \in \Phi_i^{(B)}(\mathcal{D})}{\arg\max} \; \mathcal{L}_{\text{atk}}\left(f_\theta\left(\mathcal{G}_i\left(\mathcal{D}'\right)\right), y_i\right), \tag{2}$$

where $\Phi_i^{(B)}(\mathcal{D})$ denotes the set of feasible attacks containing B injection bundles under the database schema and insertion constraints, and $\mathcal{L}_{\text{atk}}$ measures the attack outcome for i.

5 Methodology

Figure 2 provides an overview of our attack framework, which combines path selection with value optimization through RelPGD and subsequent search on unsuccessful targets.

5.1 Our Problem Setting

We consider a **target-specific white-box evasion attack on classification tasks**. For each eligible instance, the attack is constructed specifically for v_i and aims to flip its prediction to the incorrect class, rather than degrade the model's overall performance. The attacker has access to the database, its schema, the graph construction and sampling procedures, and the architecture and trained parameters θ of the victim model. The model parameters remain fixed, and the attack modifies only the target's test-time relational context through new bundle injections.

Attack objective in our setting. Let $\delta \in (0, 1)$ be the classification threshold on the predicted probability, selected on the validation set. $z_i' = f_\theta(\mathcal{G}_i')$ be the corresponding attacked logit, and clean logit be z_i. We convert the probability threshold δ to the logit scale as $\ell = \log\left(\delta/(1-\delta)\right)$ and compare the attacked logit z_i' against ℓ. For the true label $y_i \in \{0, 1\}$, let $a_i = 1 - 2y_i$. We define the attack margin as $m_i' = a_i(z_i' - \ell)$, which is positive when the attacked logit lies strictly on the incorrect side of the boundary. The optimal injection attack is defined as: $\mathbf{b}_i^* = \underset{\mathbf{b} \in \Phi_i^{(B)}(\mathcal{D})}{\arg\max} \; m_i'$.

5.2 Path Selection

Path selection identifies the schema paths where inserted rows can most effectively influence the target prediction. For each prediction instance i, we enumerate a candidate path set $\mathcal{P}$ from the database schema. Performing a complete value search over every feasible path is computationally expensive. We therefore rank the candidate paths using three signals: (i) the prediction change

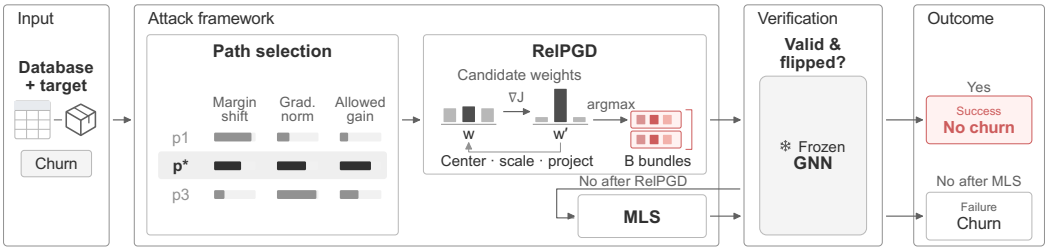

Figure 2: Overview of the RIA.

caused by an initial bundle, (ii) the sensitivity of the target prediction to changes in its inserted attributes, and (iii) the additional improvement achieved through valid attribute substitutions. Only the highest-ranked paths are considered in the subsequent value search.

Overall anchor-based path evaluation. A path specifies where an insertion attaches, but its effect also depends on the values propagated through GNN encoders and message passing, which determine both the prediction shift and local attribute gradients. We therefore use observed rows as anchor bundles to estimate these value-dependent effects before full attribute optimization, with multiple anchors reducing sensitivity to initialization. For each path $p_i \in \mathcal{P}$, we construct an anchor set $\mathcal{R}_{p_i}$ from observed fact and dimension rows that conform to the path and are available by t_i. Each anchor $r \in \mathcal{R}_{p_i}$ instantiates a single injection bundle b_{p_i}.

Each bundle is evaluated under the specified path to obtain an attacked logit z'_{p_i} and attack margin $m'_{p_i} = a_i(z'_{p_i} - \ell)$. Its deviation from the clean margin measures the initial insertion effect. At the same anchor, we compute attribute sensitivities and first-order gains over admissible substitutions. Together, these signals capture both a path's immediate effect and its potential for subsequent attribute optimization, and are used to rank candidate paths.

Insertion response. We measure the initial insertion effect by the change from the clean margin $m_i = a_i(z_i - \ell)$, $c_{p_i} = m'_{p_i} - m_i = a_i\left(z'_{p_i} - z_i\right)$:

$$\Sigma_{p_i} = \mathrm{TopMean}_{k,\, r\in\mathcal{R}_{p_i}}\, c_{p_i}. \tag{3}$$

The operator $\mathrm{TopMean}_k$ averages the largest $\min(k, |\mathcal{R}_{p_i}|)$ responses across anchors. The score captures the messages of inserted rows and the aggregation changes represented by this computation. A larger Σ_{p_i} indicates a more favorable initial shift in the attack margin.

Gradient sensitivity. Let $\mathcal{I}_p$ be the set of modifiable attribute columns in the fact and dimension rows along path p. Let $\mathbf{h}_c = \phi_c(\mathbf{x}_c)$ denote the encoded representation of column c at the input to the differentiable scorer. For each anchor bundle b_{p_i}, we compute the gradient of the attack margin with respect to its encoded attributes $\mathbf{g}_{p_i,c} = \partial m'_{p_i}/\partial \mathbf{h}_c = a_i\left(\partial z'_{p_i}/\partial \mathbf{h}_c\right)$.

To compute gradient sensitivity, we evaluate these gradients at the fixed, constraint-valid reference bundle produced by the common initialization procedure without optimizing the attack objective:

$$\Gamma_{p_i} = \sum_{c\in\mathcal{I}_p} \|\mathbf{g}_{p_i,c}\|_2 . \tag{4}$$

A larger Γ_{p_i} indicates greater local sensitivity of the attack margin to changes in the encoded attributes at the reference bundle, without accounting for the changes achievable through admissible candidate values.

Admissible directional gain. A large gradient does not necessarily imply an effective attribute substitution, since admissible values may allow only small changes or changes in unfavorable directions. Let $\mathcal{X}_{p_i,c}$ contain the candidate replacement values for attribute group $c \in \mathcal{I}_p$, and let $\mathbf{x}_{p_i,c} \in \mathcal{X}_{p_i,c}$ be its current value in anchor bundle b_{p_i}. The mapping ϕ_c converts these values into the encoded representation used to compute $\mathbf{g}_{p_i,c}$. We define $\rho_{p_i,c} = \left[\max_{\mathbf{x}'_{p_i,c}} \left\langle \mathbf{g}_{p_i,c}, \phi_c(\mathbf{x}'_{p_i,c}) - \phi_c(\mathbf{x}_{p_i,c})\right\rangle\right]_+$. Here, $[u]_+ = \max\{0, u\}$, and $\mathbf{x}'_{p_i,c}$ is a candidate

replacement for the current value $\mathbf{x}_{p_i,c}$. The gradient is evaluated at the current anchor bundle. The inner product approximates the resulting increase in the attack margin, accounting for both the attack direction and the changes available in the candidate space.

We sum the d largest gains at each anchor and then aggregate across anchors:

$$Q_{p_i} = \mathrm{TopSum}_{d,\ c\in\mathcal{I}_p}\, \rho_{p_i,c}, \qquad D_{p_i} = \mathrm{TopMean}_{k,\ r\in\mathcal{R}_{p_i}}\, Q_{p_i}. \tag{5}$$

Here, TopSum_d sums the largest $\min(d, |\mathcal{I}_p|)$ gains.

Budget-conditioned calibration. For each of the three scores, we rank the candidate paths in $\mathcal{P}_i$ and rescale the ranks to $[0, 1]$, assigning higher normalized scores to larger values and average ranks to ties. We select the path with the highest combined score:

$$S_{p_i,B} = \widetilde{\Gamma}_{p_i} + \beta_B \widetilde{\Sigma}_{p_i} + \gamma_B \widetilde{D}_{p_i}, \qquad p^*_{i,B} = \arg\max_{p_i\in\mathcal{P}} \{S_{p_i,B}\}. \tag{6}$$

For each budget B, we choose (β_B, γ_B) from a fixed nonnegative grid to maximize the number of successful attacks along the selected paths on calibration targets. We fit the coefficients on the validation split and keep them fixed when selecting paths on the test split.

5.3 Value Optimization

Each value, including each event time, is selected from a finite candidate list. All B bundles are evaluated simultaneously: the victim prediction is recomputed with the complete insertion added to the database, so the new rows contribute alongside the target's existing neighbors rather than replacing them.

Value optimization shares the heterogeneous variables and integrity requirements of constrained tabular evasion (Simonetto et al., 2022). Following CAA (Simonetto et al., 2024), we first apply the gradient attack to every target, then apply the search attack only to targets for which the gradient attack fails. We describe the two adaptations below.

Gradient attack: relational projected gradient descent. To optimize discrete candidate choices with gradients, RelPGD assigns continuously adjustable weights to the candidates in each list (Jang et al., 2017; Liu et al., 2019). For prediction instance i, consider a single inserted row and a writable column c with $n_{i,c}$ candidate values, omitting the row index for simplicity. Its weight vector $\mathbf{w}_{i,c}$ is nonnegative and sums to one. Assigning weight one to a single candidate recovers a discrete choice. Let w_i collect these weight vectors across all rows in the B injection bundles for instance i, and let $\mathcal{W}_i$ denote the space satisfying the nonnegativity and unit-sum conditions for every list. During the forward pass, the weights determine a weighted combination of the corresponding candidate representations, yielding the target logit $\widetilde{\mathbf{z}}_i(w_i)$ for the complete insertion. RelPGD maximizes an objective $J_{\mathrm{rel},i}(w_i)$ defined as the cross-entropy loss for the target's true label minus a differentiable penalty for integrity violations, including cross-row conditions.

The update preserves gradient differences between candidates within a list while balancing gradient scales across lists. Unlike CAPGD's default sign update (Simonetto et al., 2024), RelPGD retains gradient magnitudes to distinguish candidates with the same gradient sign. For each list, we center its gradient and normalize by its largest absolute centered component:

$$g_{i,c} = \nabla_{\mathbf{w}_{i,c}} J_{\mathrm{rel},i}(w_i), \qquad h_{i,c} = g_{i,c} - \frac{1}{n_{i,c}}\big(\mathbf{1}^\top g_{i,c}\big)\mathbf{1}, \qquad d_{i,c} = \frac{h_{i,c}}{\|h_{i,c}\|_\infty}. \tag{7}$$

We set $d_{i,c} = 0$ when $h_{i,c} = 0$. Centering preserves pairwise gradient differences and makes the direction sum to zero. Rescaling preserves these differences up to a common factor within each list, while giving every nonzero direction a unit maximum absolute component regardless of its raw gradient scale. Projection can still produce different displacements across lists.

Following APGD and CAPGD (Croce & Hein, 2020; Simonetto et al., 2024), we take a gradient step along the normalized directions, project the candidate weights, combine the projected step with the previous iteration's displacement using momentum, and project again. Each projection restores nonnegativity and a unit sum (Duchi et al., 2008). At progress checkpoints, stalled improvement triggers step-size halving and restoration of the best state.

At each iteration, we select the highest-weight candidate from each list and evaluate the resulting insertion using the (C1)-(C4) checks and a separate victim forward pass. We retain valid insertions without overwriting the continuous weights, allowing small weight changes to accumulate across iterations.

Search attack: multi-start local search. MLS refines discrete candidate assignments through multi-start coordinate search. We choose MLS for its low search overhead and because the finite candidate lists allow each value choice to be refined by directly comparing its alternatives while holding the remaining choices fixed. A replacement can select any listed candidate; it need not stay close to the starting value.

We draw starts from a random pool and two structured constructions to explore different initial combinations. From each start, we visit the B injection bundles and update their candidate columns in turn, comparing listed alternatives and accepting the best only if it strictly improves the attack margin.

6 Experimental Setup

Datasets, tasks, and victims. We evaluate twelve binary entity-classification tasks from seven RelBench datasets (Robinson et al., 2024) against separately trained HeteroSAGE (Hamilton et al., 2017) and RelGNN (Chen et al., 2025) victims. Because each victim has its own target set of initially correctly classified targets and ASR denominator, we do not compare them by subtracting ASRs. [1]

Baselines and budgets. Without search, we compare CPGD, CAPGD, and RelPGD; with search, we compare Random, CAA (CAPGD followed by MOEVA; Simonetto et al., 2022; 2024), and RelPGD followed by MLS. Search methods share candidates, provenance, weights, denominators, seeds, and a $10{,}000$-evaluation cap, although MLS may stop earlier. Because inserted rows have no original, we remove the baselines' ϵ-ball projections and MOEVA's distance objective while retaining all domain and integrity constraints. Additional experiments varying the insertion budget are reported in Appendix D.2.

Evaluation. We compute stratum-weighted ASR using actual-graph, sampler-included model forwards, rebuilding the graph for every candidate. Since full-frame search is infeasible at up to $10{,}000$ evaluations per target, we attack a stratified sample, weight it back to the eligible frame, and search only gradient failures. We evaluate with multiple injection budgets B while the main results focus on $B = 2$ with two gradient starts, depth 30, holding all other components fixed. For path selection, all selectors use the same RelPGD value optimizer; we compare our selector with uniform-random (**Random**) and relation-weight-norm selection (**W-norm**), and report an **Oracle** that chooses a successful evaluated path whenever one exists as an ASR upper bound. In the path-selection experiments, paths are selected individually for each target; in the value-optimization experiments, a single path is fixed for all targets within each task by majority vote over the per-target selections. Full settings including target set statistics appear in Appendix B.

7 Results

Path selection performance. Table 1 evaluates how well each scorer orders a task's admissible two-hop paths on HeteroSAGE, together with the attack success attained after selecting a path. We compare our method against W-norm, a parameter-only heuristic that assigns higher scores to paths containing relations with larger norms of the trained message-transformation weights. We also measure agreement between each predicted path ranking and the ranking induced by the realized per-path attack effect using Spearman's rank correlation ρ and Kendall's rank correlation τ.

Across tasks, our method aligns substantially better with the per-path effect than W-norm, whose rankings are often weak and occasionally inverted. This contrast highlights a fundamental limita-

[1] Path-selection experiments are conducted only on HeteroSAGE over nine multi-path tasks, since three tasks from *amazon* and *hm* each have only a single trivial path.

[2] A dash (—) marks entries where the rank correlation is undefined: every candidate path yields the same realized attack success (here, no target is flipped on any path), so there is no variation to rank.

Table 1: Path-selection performance on HeteroSAGE for $B = 2$, seed = 42. The left block reports ASR, while the right block reports Spearman's ρ and Kendall's τ correlations with the realized per-path effects.[2] The best non-oracle ASR in each row is shown in **bold**. Full results are provided in Appendix C.

Dataset	Task	Random	W-norm	Ours	Oracle	W-norm ρ/τ	Ours ρ/τ
avito	user-visits	30.3	29.9	**30.6**	32.5	-0.63 / -0.60	+0.40 / +0.33
	user-clicks	0.2	**0.3**	**0.3**	0.3	+0.78 / +0.60	+0.95 / +1.00
event	user-repeat	4.7	**15.3**	7.5	15.3	+0.95 / +1.00	+0.74 / +0.60
	user-ignore	0.0	**0.1**	**0.1**	0.1	0.00 / 0.00	0.00 / 0.00
f1	driver-dnf	14.8	8.4	**29.5**	34.9	-0.65 / -0.50	+0.87 / +0.78
	driver-top3	0.0	0.0	0.0	0.0	–	–
stack	user-engagement	39.4	**78.2**	63.4	78.3	+0.80 / +0.67	+0.80 / +0.67
	user-badge	43.7	5.9	**96.3**	99.1	-0.80 / -0.67	+0.20 / 0.00
trial	study-outcome	1.7	1.1	**2.5**	3.1	-0.20 / -0.20	-0.70 / -0.60
Macro average		15.0	15.5	**25.6**	29.3	+0.03 / +0.04	**+0.41 / +0.35**

tion of W-norm path scores: large message-weight magnitude does not necessarily imply that a path admits effective perturbations under the available candidate values and integrity constraints. Ours instead evaluates the victim's *value-dependent* response to admissible candidate-row insertions, directly accounting for whether a path can move the target margin in practice.

When our path selector helps. In most tasks where our selector exhibits substantially stronger rank correlations (ρ/τ) than W-norm, it also achieves higher attack success. However when all paths are similarly effective, the choice of path has little influence on the outcome. This implies that when no path supports an effective attack, even a perfect selector cannot substantially improve success.

Overall, while the magnitude of the gain varies with path heterogeneity and insertion budget, our method improves average path ranking over static weight magnitudes and delivers its largest benefits precisely when the choice of path has a substantial effect on attack success. Further discussions along with additional experiments, ablation studies, and detailed results, are provided in Appendix C.

Gradient attack performance. Table 2 compares value optimizers sharing the same model, targets, path, and candidate lists within each victim, with two starts and 30 optimization iterations per start. RelPGD achieves the highest mean ASR in most displayed settings, with substantial gains over CAPGD on *amazon*/item-churn and *f1*/driver-dnf. Because these gains require no changes to writable tables or connections to the target, value optimization remains consequential after structural placement is fixed. This advantage is not universal: on RelGNN's *stack*/user-badge task, CAPGD finds successful insertions more often within the same candidate space, showing that one optimizer's failure does not establish the absence of feasible attacks. On HeteroSAGE's *avito*/user-clicks and *event*/user-ignore tasks, all three methods have little success, leaving open whether stronger optimization could recover additional attacks. The contrasting RelGNN results indicate that attack effectiveness depends on the victim-specific prediction setting, although differences in eligible targets prevent attributing these contrasts solely to architecture.

Table 2: Weighted ASR of the gradient attacks for one selected task per dataset at $B = 2$ (%, ↑; mean ± SD over attack seeds 42–44). Bold marks the best mean per row, ties included. Dataset names omit the `rel-` prefix. Table 10 reports all twelve tasks.

Dataset	Task	Victim	CPGD	CAPGD	RelPGD
amazon	item-churn	HeteroSAGE	13.45 ±0.54	22.36 ±0.06	**31.55 ±0.05**
		RelGNN	17.64 ±0.34	30.17 ±0.66	**33.92 ±0.35**
avito	user-clicks	HeteroSAGE	0.01 ±0.00	0.02 ±0.00	**0.04 ±0.04**
		RelGNN	12.38 ±1.19	62.30 ±2.45	**71.51 ±1.19**
event	user-ignore	HeteroSAGE	0.00 ±0.00	**0.07 ±0.00**	**0.07 ±0.00**
		RelGNN	16.12 ±1.22	32.40 ±0.58	**48.23 ±3.01**
f1	driver-dnf	HeteroSAGE	10.09 ±1.06	27.81 ±0.84	**35.10 ±0.61**
		RelGNN	24.53 ±1.59	37.63 ±0.71	**52.96 ±0.20**
hm	user-churn	HeteroSAGE	7.65 ±0.01	24.25 ±0.05	**30.54 ±1.37**
		RelGNN	33.25 ±3.40	83.70 ±1.59	**87.74 ±1.45**
stack	user-badge	HeteroSAGE	2.68 ±0.09	4.83 ±0.31	**5.01 ±0.30**
		RelGNN	24.24 ±1.31	**90.37 ±0.45**	63.01 ±1.19
trial	study-outcome	HeteroSAGE	1.18 ±0.11	2.74 ±0.00	**2.81 ±0.11**
		RelGNN	2.58 ±0.04	**3.91 ±0.11**	3.52 ±0.00

Table 3: RelPGD ablation on HeteroSAGE: numeric/time candidate weights and the scaled gradient (Eq. 7). Task-averaged ASR (%) across twelve tasks, mean ± SD over three attack seeds.

RelPGD variant	Mean ASR ↑
RelPGD (w/ both)	$\mathbf{19.73 \pm 0.16}$
w/o numeric/time weights	18.07 ± 0.11
w/o scaled gradient	16.95 ± 0.23
w/o both	17.78 ± 0.08

Table 4: Per-target optimization cost for gradient attacks and search on HeteroSAGE across all evaluation targets from the twelve tasks. Times are in seconds; search time excludes scorer execution, while total time includes it. Bold marks the minimum in each cost column.

Gradient optimization			Search optimization		
Method	Backward calls ↓	Total time ↓	Method	Search time ↓	Total time ↓
CPGD	60.00	1.17	MOEVA	6.51	102.2
CAPGD	54.38	1.17	MLS (4 starts)	**0.35**	**87.4**
RelPGD	**51.81**	**0.71**	MLS (1,000 starts)	0.64	94.2

Contribution of the optimization components. Table 3 examines two RelPGD components: candidate weights for numerical and temporal choices, and a scaled direction that retains relative gradient magnitudes. The full method achieves the highest task-averaged ASR, but the representation's benefit depends on the direction: candidate weights improve ASR under scaled updates and reduce it under sign updates. This supports their combined use; extending candidate weights alone does not explain RelPGD's advantage.

Gradient attacks with search. Figure 3 shows that, with up to 10,000 additional candidate evaluations, both search methods recover substantial success on gradient failures and largely close the CAPGD–RelPGD gap: the MLS combinations reach identical final ASR, while the MOEVA combinations finish at similar levels. The gradient-stage ranking therefore need not persist after search, although these gains require additional computation and are not equal-cost improvements over gradient optimization alone. Any runtime saving depends on the number and search cost of the remaining targets and cannot be inferred from aggregate ASR alone.

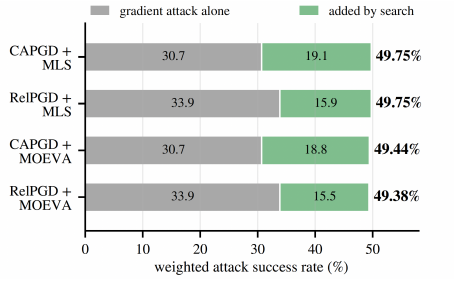

Figure 3: Contributions of gradient attacks and subsequent search on *amazon*/item-churn with RelGNN ($B = 2$, attack seed 42).

Optimization cost. Table 4 reports per-target costs on HeteroSAGE at $B = 2$. RelPGD requires fewer backward passes and less wall-clock time than gradient baselines, supporting its use as an initial attack before larger search budgets. CAPGD uses fewer backward calls than CPGD but shows no runtime reduction, indicating substantial non-gradient overhead. For search, we separate *search logic time* from total wall-clock time including scorer execution. MLS greatly reduces search logic time over MOEVA, but only modestly reduces total time, which remains dominated by scorer execution. Increasing MLS from four to 1,000 starts further increases total time without evidence of improved ASR.

8 Conclusion and Limitations

We showed that schema-valid injections can affect RDL predictions when existing records and model parameters remain unchanged. Their effectiveness depends jointly on the injection path and admissible values: the former determines how injected evidence reaches the target, while the latter determines how strongly it shifts the prediction. Our framework improves path selection over static weight-based scores and value optimization over adapted tabular-attack baselines. These results show that preserving database integrity alone does not ensure prediction robustness: valid new records can still introduce adversarial evidence through permitted relational connections. RDL robustness evaluations should therefore include permitted row insertions as an attack surface.

We study append-only test-time attacks against frozen white-box victims using finite candidate sets and declared constraints. Poisoning, unrestricted value synthesis, and persistent operational effects remain outside scope. Future work could examine attacks under partial model knowledge and defenses that limit the influence of newly inserted records while preserving the predictive value of legitimate relational updates. Appendix E provides further discussion.

ETHICS STATEMENT

This work studies the vulnerability of relational deep learning models to adversarial row insertion. The experiments are conducted on benchmark datasets and local model instances, without modifying operational databases or targeting deployed services. Our threat model explicitly limits the attacker to declared write permissions and requires inserted records to satisfy the specified integrity constraints. These assumptions describe the capabilities evaluated in our experiments, not authorization to attack real systems. Although the findings can support more robust relational learning systems, the proposed techniques could be misused to manipulate predictions. We therefore emphasize controlled evaluation and authorized security testing. Public availability of a dataset does not eliminate privacy concerns; any release of experimental artifacts will respect the applicable dataset licenses and access restrictions.

REPRODUCIBILITY STATEMENT

To support reproducibility, we will publicly release the source code and configurations used in our experiments upon publication. The release will include the implementation of path selection and value optimization, scripts for preparing the benchmark datasets and victim models, and code for attack evaluation. We will document the insertion constraints, candidate construction, attack budgets, sampling and evaluation protocols, and random seeds. Instructions will specify the software dependencies and hardware used, together with the commands needed to reproduce the main results. Datasets will be obtained through their original distribution channels, subject to their licenses and access requirements.

AI USE STATEMENT

AI tools assisted with manuscript drafting and revision, including improving clarity, organization, terminology, and mathematical notation. They were also used to examine methodological assumptions, identify potential inconsistencies between the manuscript and implementation, and assist with code development and experimental analysis. The authors are responsible for verifying the resulting text, code, and analyses, and for the accuracy and integrity of the reported findings.

REFERENCES

Tianlang Chen, Charilaos Kanatsoulis, and Jure Leskovec. RelGNN: Composite message passing for relational deep learning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8296–8312. PMLR, 2025. URL https://proceedings.mlr.press/v267/chen25ad.html.

Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 2206–2216. PMLR, 2020. URL https://proceedings.mlr.press/v119/croce20b.html.

Hanjun Dai, Hui Li, Tian Tian, Xin Huang, Lin Wang, Jun Zhu, and Le Song. Adversarial attack on graph structured data. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1115–1124. PMLR, 2018. URL https://proceedings.mlr.press/v80/dai18b.html.

John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the ℓ_1-ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning*, pp. 272–279. Association for Computing Machinery, 2008. doi: 10.1145/1390156.1390191. URL https://doi.org/10.1145/1390156.1390191.

Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Position: Relational deep learning - graph representation learning on relational databases. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 13592–13607. PMLR, 2024. URL https://proceedings.mlr.press/v235/fey24a.html.

Alan Gany, Bogdan Cautis, and Silviu Maniu. Structural adversarial attacks on relational deep learning under integrity constraints. arXiv preprint arXiv:2607.07089, 2026. URL https://arxiv.org/abs/2607.07089.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://papers.nips.cc/paper/6703-inductive-representation-learning-on-large-graphs.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1611.01144.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan E. Lenssen, Yiwen Yuan, Zecheng Zhang, Xinwei He, and Jure Leskovec. RelBench: A benchmark for deep learning on relational databases. In *Advances in Neural Information Processing Systems*, volume 37, pp. 21330–21341. Curran Associates, Inc., 2024. doi: 10.52202/079017-0672. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/25cd345233c65fac1fec0ce61d0f7836-Abstract-Datasets_and_Benchmarks_Track.html.

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, 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, pp. 27817–27849. Curran Associates, Inc., 2024. doi: 10.52202/079017-0873. URL

https://proceedings.neurips.cc/paper_files/paper/2024/hash/3106a1387b24c33bf66682e7ff393d56-Abstract-Conference.html.

Xiaoyun Wang, Minhao Cheng, Joe Eaton, Cho-Jui Hsieh, and S. Felix Wu. Fake node attacks on graph convolutional networks. *Journal of Computational and Cognitive Engineering*, 1(4): 165–173, 2022. doi: 10.47852/bonviewJCCE2202321.

Xu Zou, Qinkai Zheng, Yuxiao Dong, Xinyu Guan, Evgeny Kharlamov, Jialiang Lu, and Jie Tang. TDGIA: Effective injection attacks on graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2461–2471. Association for Computing Machinery, 2021. doi: 10.1145/3447548.3467314. URL https://doi.org/10.1145/3447548.3467314.

Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. Adversarial attacks on neural networks for graph data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2847–2856. Association for Computing Machinery, 2018. doi: 10.1145/3219819.3220078. URL https://doi.org/10.1145/3219819.3220078.

A Benchmark Details

We evaluate relational injection attacks on twelve binary entity classification tasks from seven RelBench databases (Robinson et al., 2024). These databases cover a range of application domains, including e-commerce, online marketplaces, social platforms, motorsports, and clinical research. We exclude regression and recommendation tasks because our attack formulation considers target-specific label flips in binary classification.

A.1 Task statistics

Table 5: Statistics of the binary entity classification tasks used in our evaluation. Overlap denotes the percentage of target entities that occur in both the training and test splits.

Database	Task	Train	Validation	Test	Entity overlap (%)
rel-amazon	user-churn	4,732,555	409,792	351,885	88.0
	item-churn	2,559,264	177,689	166,842	93.1
rel-avito	user-clicks	59,454	21,183	47,996	45.3
	user-visits	86,619	29,979	36,129	64.6
rel-event	user-repeat	3,842	268	246	11.5
	user-ignore	19,239	4,185	4,010	21.1
rel-f1	driver-dnf	11,411	566	702	50.0
	driver-top3	1,353	588	726	50.0
rel-hm	user-churn	3,871,410	76,556	74,575	89.7
rel-stack	user-engagement	1,360,850	85,838	88,137	97.4
	user-badge	3,386,276	247,398	255,360	96.9
rel-trial	study-outcome	11,994	960	825	0.0

A.2 Databases and classification tasks

All tasks considered in this work are binary entity classification tasks. For each target entity, the label indicates whether a specified event occurs within a fixed prediction horizon following the prediction time.

rel-amazon. This database contains Amazon users, products, reviews, and product metadata. Its relational structure connects users to products through their reviews and includes numerical, categorical, and textual attributes such as ratings, product categories, prices, and review content.

- user-churn: Predict whether a user will cease posting product reviews during the following three months.

- `item-churn`: Predict whether a product will receive no new reviews during the following three months.

rel-avito. This database is derived from an online marketplace and contains users, advertisements, search activity, and interaction records. Its tables describe both the properties of individual advertisements and the ways in which users view and click them.

- `user-clicks`: Predict whether a user will click multiple advertisements during the next four days.
- `user-visits`: Predict whether a user will visit multiple advertisements during the next four days.

rel-event. This database records user activity on a social event platform. It includes users, events, invitations, attendance responses, and social connections, allowing prediction from both prior user behavior and relations among users and events.

- `user-repeat`: Predict whether a user who recently attended an event will attend another event during the next seven days.
- `user-ignore`: Predict whether a user will ignore more than two event invitations during the next seven days.

rel-f1. This database contains historical Formula 1 records, including drivers, constructors, races, qualifying sessions, circuits, and race outcomes. Its schema captures the relationships among drivers, teams, events, and their competitive results.

- `driver-dnf`: Predict whether a driver will fail to finish a race during the next month.
- `driver-top3`: Predict whether a driver will achieve a top-three qualifying result during the next month.

rel-hm. This e-commerce database contains customers, products, and purchase transactions from H&M. Customer characteristics and product descriptions are connected through time-stamped transaction records.

- `user-churn`: Predict whether a customer will make no purchase during the following week.

rel-stack. This database represents activity on the Stack Exchange network. It contains users, posts, comments, votes, and badges, thereby capturing multiple forms of participation and recognition within the platform.

- `user-engagement`: Predict whether a user will post, comment, or vote during the next three months.
- `user-badge`: Predict whether a user will receive a new badge during the next three months.

rel-trial. This database is constructed from clinical trial records and contains studies, sponsors, facilities, interventions, eligibility criteria, and reported outcomes. It represents the organizational and clinical context associated with each trial.

- `study-outcome`: Predict whether a clinical trial will achieve its primary outcome during the following year.

B Evaluation Protocol

B.1 Target set statistics

We evaluate twelve tasks from seven RelBench datasets. Table 6 reports the target set statistics for HeteroSAGE and RelGNN. A target is an $(\text{entity}, \text{prediction time})$ pair. Each victim has its own checkpoint, eligibility decisions, and target set; the reported counts therefore do not transfer between victims.

Targets are sampled with seed 42 by stratifying jointly by binary label and degree. The degree bins are 0, 1–2, 3–8, 9–32, 33–127, and 128+, with at most 80 targets sampled per occupied cell. *Strata* reports the number of occupied label–degree cells, rather than the number of distinct sampling weights. For each victim and task, attack seeds 42, 43, and 44 reuse the same target set and vary only the attack randomness.

For HeteroSAGE, the saved sampling frame contains only targets classified correctly by the clean victim. The reported sample consists of targets retained after preparation filters and covers every occupied stratum in the eligible frame. ASR retains the saved population denominator and the original sampling weights, without renormalization after preparation filtering.

For RelGNN, targets are drawn from the saved historical sampling frame and subsequently audited with the clean victim. The reported sample includes targets found to be incorrectly classified during this audit; only clean-correct targets are eligible for attack. The ASR denominator is the sum of the sampling weights of the audited clean-correct targets. Denominators are rounded to the nearest integer for display, whereas ASR uses their unrounded values.

Table 6: Target set statistics for HeteroSAGE and RelGNN. *Frame* denotes the sampling population; *Sample*, the sample size; *Strata*, the number of occupied label–degree cells; and *Denom.*, the ASR denominator. Model-specific sampling and denominator definitions are given above. RelGNN denominators are rounded for display only.

		HeteroSAGE				RelGNN			
Dataset	Task	Frame	Sample	Strata	Denom.	Frame	Sample	Strata	Denom.
amazon	item-churn	125,639	732	10	125,639	125,819	747	10	125,809
	user-churn	238,175	758	10	238,175	240,936	754	10	240,927
avito	user-clicks	46,404	537	9	46,404	46,448	518	8	46,443
	user-visits	28,896	725	11	28,896	28,820	776	11	28,626
event	user-ignore	1,715	211	4	1,715	1,712	212	4	1,691
	user-repeat	167	158	3	167	179	158	2	178
f1	driver-dnf	494	333	6	494	516	349	6	503
	driver-top3	580	320	6	580	628	364	6	625
hm	user-churn	57,733	638	10	57,733	54,884	674	10	54,848
stack	user-badge	246,558	826	12	246,558	245,927	755	11	245,927
	user-engagement	85,218	661	10	85,218	85,262	666	10	85,262
trial	study-outcome	543	495	8	543	523	473	8	523

B.2 Evaluation metric

We report stratum-weighted attack success rate (ASR) over targets that the clean victim classifies correctly. An attack succeeds if the inserted rows satisfy all insertion constraints and cause an incorrect prediction under the victim's validation-fixed decision threshold. Using the attack margin m'_i defined in Section 5.1, we compute

$$\widehat{\mathrm{ASR}}(B) = \frac{1}{N_{\mathrm{eligible}}} \sum_{i \in \mathcal{S}} w_i \, \mathbf{1}[A_i \text{ satisfies (C1)–(C4)} \ \wedge \ m'_i(A_i) > 0] \,, \tag{8}$$

where $\mathcal{S}$ is the sampled set of clean-correct targets, w_i is the stratum sampling weight, and N_{eligible} is the corresponding ASR denominator described above. Success is verified by evaluating the victim on the attacked graph; ties and invalid or failed constructions count as unsuccessful. For cascaded attacks, search is applied only to gradient-stage failures, while the original sampling weights and denominator are retained.

B.3 Implementation details

Experiments were conducted on Linux systems equipped with NVIDIA A100 and H100 GPUs. The HeteroSAGE pipeline used PyTorch 2.6.0, PyTorch Geometric 2.7.0, and RelBench 3.0.1. The RelGNN pipeline used PyTorch 2.4.1, PyTorch Geometric 2.8.0.post1, and RelBench 1.1.0. Both environments used PyTorch builds with CUDA 12.4. Unless otherwise specified, results are reported

as the mean and sample standard deviation over three attack seeds (42, 43, and 44), with the victim checkpoint and target set held fixed.

C Extended experiment results for path selection

We examine how the choice of an insertion path affects the outcome of a fixed value optimizer. The comparison covers nine tasks with multiple candidate paths, HeteroSAGE, and insertion budgets $B \in \{1, 2, 4, 8\}$. We first report task-level variation over seeds 42, 43, and 44, then ablate the three path scores.

C.1 Full results

Tables 7 and 8 report the task-level results. For each target, Random selects uniformly among the available paths; its reported success is the exact expectation over that selection, rather than a single sampled path. W-norm selects using the norm of the carrier relation's first-layer weight matrix. Ours is the saved capacity selector. Let I, G, and D denote insertion response, gradient sensitivity, and directional gain, respectively. After within-target rank normalization, the saved selector uses

$$S_{p_i,B} = \widetilde{\Gamma}_{p_i} + \beta_B \widetilde{\Sigma}_{p_i} + \gamma_B \widetilde{D}_{p_i} \qquad (\beta_B, \gamma_B) = \begin{cases} (2,2), & B = 1, \\ (1,2), & B = 2, \\ (1,0), & B = 4, \\ (2,2), & B = 8. \end{cases} \tag{9}$$

Table 7: Task-level ASR (%) for budgets $B = 1$ and 2. Results are mean ± population SD across seeds 42, 43, and 44. Macro rows average the nine task means. **Bold** denotes the highest displayed mean among the three selectors, including ties.

		$B = 1$			$B = 2$		
Dataset	Task	Random	W-norm	Ours	Random	W-norm	Ours
avito	user-visits	28.21 ± 0.07	29.43 ± 0.09	$\mathbf{30.17 \pm 0.63}$	$\mathbf{30.69 \pm 0.30}$	30.07 ± 0.13	30.64 ± 0.15
	user-clicks	0.05 ± 0.04	$\mathbf{0.09 \pm 0.09}$	$\mathbf{0.09 \pm 0.09}$	0.14 ± 0.07	0.24 ± 0.10	$\mathbf{0.31 \pm 0.17}$
event	user-repeat	2.25 ± 0.13	$\mathbf{7.23 \pm 0.31}$	4.37 ± 0.54	4.76 ± 0.07	$\mathbf{15.97 \pm 0.54}$	7.49 ± 1.03
	user-ignore	$\mathbf{0.10 \pm 0.13}$	0.02 ± 0.03	0.00 ± 0.00	0.30 ± 0.39	0.02 ± 0.03	$\mathbf{0.78 \pm 1.06}$
f1	driver-dnf	4.18 ± 0.35	0.13 ± 0.18	$\mathbf{8.55 \pm 0.16}$	15.27 ± 0.41	12.71 ± 3.19	$\mathbf{28.52 \pm 0.93}$
	driver-top3	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	$\mathbf{0.00 \pm 0.00}$	0.04 ± 0.05	$\mathbf{0.06 \pm 0.08}$	$\mathbf{0.06 \pm 0.08}$
stack	user-engagement	6.50 ± 0.65	6.85 ± 0.50	$\mathbf{6.95 \pm 1.11}$	39.16 ± 0.72	$\mathbf{78.36 \pm 3.31}$	53.15 ± 17.63
	user-badge	25.30 ± 1.62	2.13 ± 0.98	$\mathbf{67.02 \pm 3.62}$	42.85 ± 0.82	4.10 ± 1.28	$\mathbf{88.24 \pm 7.11}$
trial	study-outcome	1.16 ± 0.02	0.95 ± 0.00	$\mathbf{1.81 \pm 0.10}$	1.58 ± 0.06	1.13 ± 0.00	$\mathbf{2.34 \pm 0.16}$
Macro mean		7.53	5.20	**13.22**	14.98	15.85	**23.50**

C.2 Ablation for path selection

We ablate the three path-selection scores: structural response (I), gradient sensitivity (G), and directional gain (D). Table 9 enumerates the seven nonempty subsets, assigning unit weight to each enabled rank-normalized score. The pairwise combinations therefore provide leave-one-score-out comparisons against the equal-weight three-score combination. For the all-off reference, we use uniform Random selection rather than resolving identical scores through an arbitrary path ordering.

The final row reports Ours with the archived budget-specific weights, where (β_B, γ_B) equals $(2, 2)$, $(1, 2)$, $(1, 0)$, and $(2, 2)$ for $B = 1, 2, 4, 8$, respectively. Thus, at $B = 4$, the saved selector assigns zero weight to D. All variants reuse the same path-specific RelPGD search outcomes. This ablation uses seed 42 only and reports search-scorer success, rather than independently adjudicated actual-forward success.

At $B = 2$, adding D to equal-weight $I + G$ raises macro search success from 19.7% to 23.9%, a 4.2-point difference. Adding I to $G + D$ instead changes it from 24.5% to 23.9%, and adding G to $I + D$ changes it from 20.5% to 23.9%. Consequently, the contribution of a score depends on the other enabled terms.

Table 8: Task-level ASR (%) for budgets $B = 4$ and 8. Results are mean $\pm$ population SD across seeds 42, 43, and 44. Macro rows average the nine task means. **Bold** denotes the highest displayed mean among the three selectors, including ties.

		$B = 4$			$B = 8$		
Dataset	Task	Random	W-norm	Ours	Random	W-norm	Ours
avito	user-visits	32.20 ± 0.39	31.87 ± 0.25	$\mathbf{33.15 \pm 1.34}$	34.34 ± 0.58	32.10 ± 0.01	$\mathbf{35.17 \pm 3.04}$
	user-clicks	0.23 ± 0.13	$\mathbf{0.40 \pm 0.28}$	$\mathbf{0.40 \pm 0.28}$	0.33 ± 0.17	0.44 ± 0.27	$\mathbf{0.58 \pm 0.27}$
event	user-repeat	9.32 ± 0.14	$\mathbf{30.96 \pm 0.77}$	14.15 ± 1.05	13.25 ± 0.83	$\mathbf{40.88 \pm 3.50}$	21.00 ± 1.93
	user-ignore	$\mathbf{0.39 \pm 0.53}$	0.02 ± 0.03	0.38 ± 0.54	1.11 ± 0.81	0.12 ± 0.03	$\mathbf{1.38 \pm 1.67}$
f1	driver-dnf	21.08 ± 1.53	30.68 ± 6.35	$\mathbf{32.32 \pm 0.66}$	21.68 ± 1.91	$\mathbf{36.96 \pm 1.32}$	33.52 ± 4.57
	driver-top3	0.09 ± 0.06	$\mathbf{0.12 \pm 0.08}$	$\mathbf{0.12 \pm 0.08}$	0.45 ± 0.12	$\mathbf{0.59 \pm 0.17}$	$\mathbf{0.59 \pm 0.17}$
stack	user-engagement	92.23 ± 0.99	$\mathbf{98.84 \pm 0.01}$	85.78 ± 3.95	98.85 ± 0.01	98.84 ± 0.01	$\mathbf{98.86 \pm 0.01}$
	user-badge	81.16 ± 1.36	55.30 ± 3.41	$\mathbf{97.96 \pm 0.48}$	97.48 ± 0.30	94.41 ± 0.82	$\mathbf{99.06 \pm 0.01}$
trial	study-outcome	3.16 ± 0.16	2.25 ± 0.00	$\mathbf{5.02 \pm 0.77}$	4.97 ± 0.21	3.70 ± 0.02	$\mathbf{7.35 \pm 0.46}$
Macro mean		26.65	27.83	**29.92**	30.27	**34.23**	33.06

Table 9: Score-subset ablation: macro ASR (%) over nine tasks, seed 42. A "1" enables the corresponding rank-normalized score with unit weight; "0" disables it. Ours uses the budget-specific weights described above. **Bold** denotes the highest displayed value in each budget column.

Selector	I	G	D	$B = 1$	$B = 2$	$B = 4$	$B = 8$
Uniform Random	0	0	0	7.4	15.0	26.5	30.0
I	1	0	0	12.6	20.4	30.6	**34.5**
G	0	1	0	10.5	20.9	30.0	32.2
D	0	0	1	12.5	25.2	31.2	32.3
$I + G$	1	1	0	11.7	19.7	30.0	32.5
$I + D$	1	0	1	12.5	20.5	29.9	32.5
$G + D$	0	1	1	11.8	24.5	30.4	31.7
$I + G + D$	1	1	1	12.8	23.9	**31.4**	32.8
Ours (budget-weighted)	β_B	1	γ_B	**12.9**	**25.6**	30.0	32.5

D Extended experiment results for value optimization

D.1 Full results

Table 10 reports results across all twelve tasks and both victims. Additional search often narrows the gap between CAPGD and RelPGD, although the gains vary across tasks.

D.2 Effect of insertion budget

Figure 4 shows that the effect of the insertion budget is task dependent: some tasks plateau at moderate budgets, while others show substantial gains only at larger budgets. RelPGD often achieves higher ASR at smaller budgets, but its advantage over CAPGD can shrink or reverse as the budget increases. The non-monotonic behavior of CPGD on some tasks also shows that a larger insertion allowance does not necessarily yield a more successful attack.

E Limitations

We evaluate target-specific test-time attacks against frozen white-box victims under declared insertion permissions and integrity constraints. Our findings are conditional on these knowledge and access assumptions; poisoning, retraining, and post-attack adaptation remain outside scope. Search is limited to finite candidate sets and does not include unrestricted value or text synthesis. Path

Table 10: Weighted ASR of gradient attacks and search-based methods at $B = 2$ (%, ↑; mean ± SD over attack seeds 42–44). C and R denote CAPGD and RelPGD, respectively; M and L denote MOEVA and multi-start local search (MLS). A "+" indicates that search is applied only to targets on which the gradient attack fails. Bold marks the highest mean within each row and column group, including ties. Dataset names omit the `rel-` prefix.

			Gradient only			With search				
Dataset	Task	Victim	CPGD	CAPGD	RelPGD	Random	C+M	C+L	R+M	R+L
amazon	user-churn	HeteroSAGE	45.42 ± 1.42	71.44 ± 0.25	**74.40 ± 0.88**	74.74 ± 0.10	76.19 ± 0.14	**77.28 ± 0.15**	76.19 ± 0.14	**77.28 ± 0.15**
		RelGNN	40.86 ± 1.21	65.79 ± 0.66	**76.37 ± 0.16**	73.49 ± 0.80	79.97 ± 0.16	**80.22 ± 0.07**	79.97 ± 0.16	**80.22 ± 0.07**
	item-churn	HeteroSAGE	13.45 ± 0.54	22.36 ± 0.06	**31.55 ± 0.05**	28.34 ± 0.49	33.65 ± 0.59	**35.72 ± 0.04**	33.71 ± 0.52	**35.72 ± 0.04**
		RelGNN	17.64 ± 0.34	30.17 ± 0.66	**33.92 ± 0.35**	37.13 ± 0.27	49.96 ± 0.53	**50.07 ± 0.54**	49.82 ± 0.55	**50.07 ± 0.53**
avito	user-visits	HeteroSAGE	29.27 ± 0.03	29.97 ± 0.01	**29.99 ± 0.00**	30.01 ± 0.01	30.04 ± 0.00	**30.06 ± 0.00**	30.05 ± 0.00	**30.06 ± 0.00**
		RelGNN	28.63 ± 0.37	**29.89 ± 0.01**	29.87 ± 0.02	30.41 ± 0.03	30.54 ± 0.04	**30.56 ± 0.03**	30.54 ± 0.04	**30.56 ± 0.03**
	user-clicks	HeteroSAGE	0.01 ± 0.00	0.02 ± 0.00	**0.04 ± 0.04**	**0.09 ± 0.00**	**0.09 ± 0.00**	**0.09 ± 0.00**	**0.09 ± 0.00**	**0.09 ± 0.00**
		RelGNN	12.38 ± 1.19	62.30 ± 2.45	**71.51 ± 1.19**	75.08 ± 2.42	96.80 ± 0.26	**99.60 ± 0.18**	96.80 ± 0.26	**99.60 ± 0.18**
event	user-repeat	HeteroSAGE	**1.98 ± 0.00**	**1.98 ± 0.00**	**1.98 ± 0.00**	**1.98 ± 0.00**	**1.98 ± 0.00**	**1.98 ± 0.00**	**1.98 ± 0.00**	**1.98 ± 0.00**
		RelGNN	3.52 ± 0.00	**4.79 ± 0.00**	**4.79 ± 0.00**	4.72 ± 0.00	**4.79 ± 0.00**	**4.79 ± 0.00**	**4.79 ± 0.00**	**4.79 ± 0.00**
	user-ignore	HeteroSAGE	0.00 ± 0.00	**0.07 ± 0.00**	**0.07 ± 0.00**	**0.07 ± 0.00**	**0.07 ± 0.00**	**0.07 ± 0.00**	**0.07 ± 0.00**	**0.07 ± 0.00**
		RelGNN	16.12 ± 1.22	32.40 ± 0.58	**48.23 ± 3.01**	9.03 ± 0.67	**55.16 ± 1.77**	52.90 ± 2.82	**55.16 ± 1.77**	53.31 ± 3.38
f1	driver-dnf	HeteroSAGE	10.09 ± 1.06	27.81 ± 0.84	**35.10 ± 0.61**	28.03 ± 0.97	36.01 ± 0.24	**36.15 ± 0.00**	**36.15 ± 0.00**	**36.15 ± 0.00**
		RelGNN	24.53 ± 1.59	37.63 ± 0.71	**52.96 ± 0.20**	19.39 ± 1.34	51.56 ± 0.47	**58.16 ± 0.69**	57.36 ± 0.00	**58.16 ± 0.69**
	driver-top3	HeteroSAGE	2.24 ± 0.17	2.41 ± 0.00	**2.76 ± 0.00**	**2.76 ± 0.00**	**2.76 ± 0.00**	**2.76 ± 0.00**	**2.76 ± 0.00**	**2.76 ± 0.00**
		RelGNN	0.00 ± 0.00	0.00 ± 0.00	**0.29 ± 0.25**	**3.85 ± 0.22**	2.70 ± 0.22	3.10 ± 0.00	2.70 ± 0.22	3.10 ± 0.00
hm	user-churn	HeteroSAGE	7.65 ± 0.01	24.25 ± 0.05	**30.54 ± 1.37**	24.71 ± 0.51	41.44 ± 1.00	**44.45 ± 0.17**	41.45 ± 1.00	**44.45 ± 0.17**
		RelGNN	33.25 ± 3.40	83.70 ± 1.59	**87.74 ± 1.45**	89.75 ± 0.29	90.92 ± 0.02	**91.01 ± 0.08**	90.92 ± 0.02	**91.01 ± 0.08**
stack	user-engagement	HeteroSAGE	14.52 ± 0.54	21.69 ± 0.36	**22.61 ± 0.45**	22.71 ± 0.11	**25.55 ± 0.22**	25.44 ± 0.03	**25.55 ± 0.22**	25.43 ± 0.03
		RelGNN	32.93 ± 3.42	80.63 ± 1.47	**97.12 ± 0.08**	86.50 ± 0.67	98.77 ± 0.27	98.73 ± 0.35	**98.92 ± 0.00**	98.73 ± 0.35
	user-badge	HeteroSAGE	2.68 ± 0.09	4.83 ± 0.31	**5.01 ± 0.30**	4.74 ± 0.01	6.50 ± 0.00	**6.63 ± 0.09**	6.50 ± 0.00	**6.63 ± 0.09**
		RelGNN	24.24 ± 1.31	**90.37 ± 0.45**	63.01 ± 1.19	67.89 ± 1.48	97.56 ± 0.12	**97.66 ± 0.00**	97.56 ± 0.12	**97.66 ± 0.00**
trial	study-outcome	HeteroSAGE	1.18 ± 0.11	2.74 ± 0.00	**2.81 ± 0.11**	**2.93 ± 0.00**	**2.93 ± 0.00**	**2.93 ± 0.00**	**2.93 ± 0.00**	**2.93 ± 0.00**
		RelGNN	2.58 ± 0.04	**3.91 ± 0.11**	3.52 ± 0.00	3.73 ± 0.23	**4.17 ± 0.00**	**4.17 ± 0.00**	**4.17 ± 0.00**	**4.17 ± 0.00**

capacity is a linearized surrogate, and neither path selection nor value optimization guarantees an optimal attack. Consequently, failure to find a successful insertion does not certify robustness, even within the evaluated candidate space.

Variation across attack seeds is measured with each victim model and target cohort held fixed, so it does not capture variability across training seeds, cohorts, or write permissions. Differences in eligible cohorts also prevent attributing contrasts between victims solely to architecture. Because attacks are evaluated individually, interactions among jointly deployed attacks, persistent effects, detection, and mitigation remain unexamined. Compliance with the insertion constraints therefore does not establish that an attack would evade detection or remain effective after subsequent database updates or model retraining.

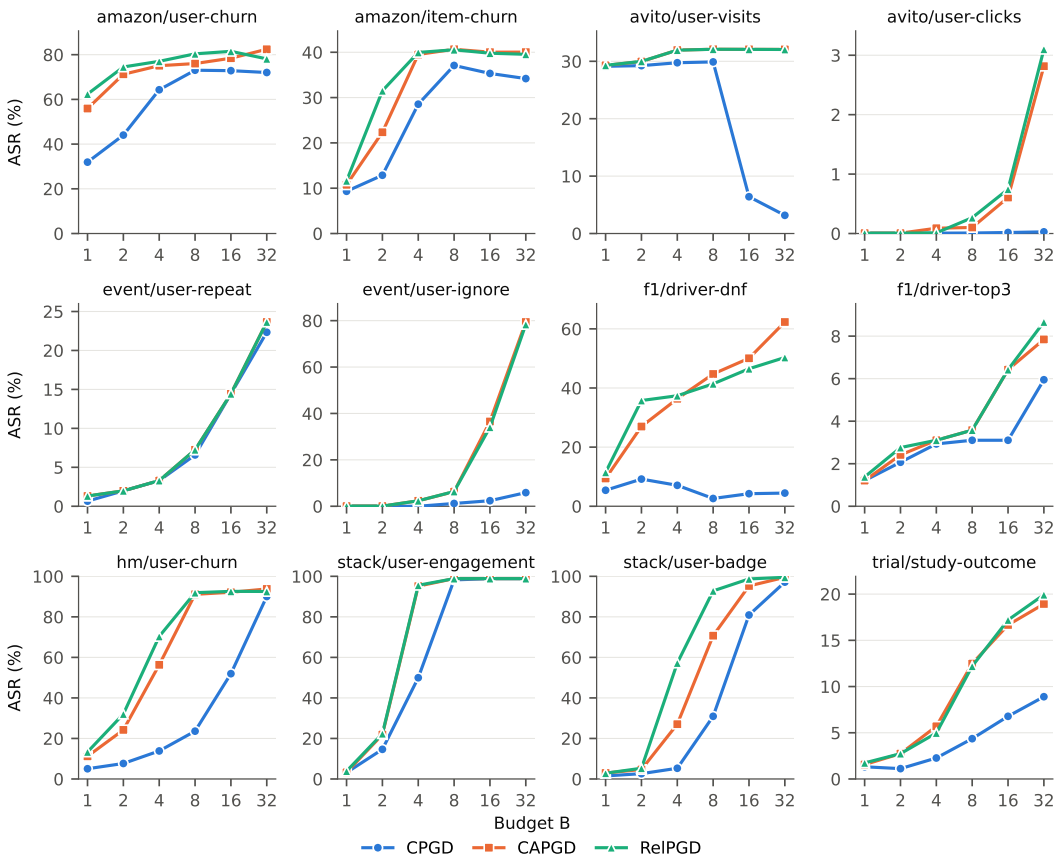

Figure 4: Weighted ASR of gradient attacks across insertion budgets on HeteroSAGE (attack seed 42). Each panel corresponds to one task.