DARC: DEPENDENCY-AWARE RELATIONAL COMPILER

Anonymous authors
Paper under double-blind review

ABSTRACT

Relational feature compilers summarize records linked to a prediction target, but column-wise aggregates cannot represent dependencies among source attributes. We introduce **DARC**, a dependency-aware relational compiler that first constructs a compact point-in-time representation from column-wise aggregates and then augments it with features derived from empirical functional dependencies. Because such features are not uniformly useful, DARC selects them on temporal folds within the training split and decides per task whether to include them at all. DARC activates dependency-aware features on 14 tasks and retains a purely aggregate representation on the rest. DARC achieves positive win–loss margins against budget-matched Canonical DFS, JUICE, and FastProp; the selected representation also transfers across six downstream predictors. Controlled collisions and real-data rewiring show that dependency features can distinguish histories with identical conventional aggregates.

1 INTRODUCTION

Recent approaches to relational prediction broadly follow two directions. Graph-based approaches learn representations directly over the linked structure of relational databases using graph message passing, relational attention, or related architectures (Fey et al., 2024; Chen et al., 2025; Wang et al., 2025; Dwivedi et al., 2026; Ranjan et al., 2026; Fey et al., 2025; Hudovernik et al., 2026). In contrast, *feature-compilation* approaches transform relational information into a fixed-length representation that can be consumed by a standard downstream predictor, as in Deep Feature Synthesis (DFS) (Kanter & Veeramachaneni, 2015) and more recent systems such as JUICE (Xu et al., 2026).

We focus on the feature-compilation direction. This formulation decouples relational representation construction from the downstream predictor, allowing the same compiled representation to be paired with a wide range of tabular models. The emergence of tabular foundation models has made this paradigm substantially more powerful by enabling strong prediction directly from compiled tabular representations (Hollmann et al., 2025; Qu et al., 2025). Recent systems such as JUICE show that combining fixed-length relational representations with in-context tabular prediction can achieve performance competitive with learned relational encoders (Xu et al., 2026). However, the effectiveness of the resulting predictor depends critically on what information is preserved during compilation. In particular, indiscriminately incorporating heterogeneous columns can hurt performance when their relevance cannot be assessed without label information (Xu et al., 2026).

Relational databases, however, provide a useful schema-level signal for deciding what to compile: functional dependencies. A functional dependency $X \to A$ states that the value of attribute set X determines the value of attribute A. Such dependencies are fundamental to relational normalization, where schemas are decomposed according to dependency structure to reduce redundancy and update anomalies (Silberschatz et al., 2019). Consequently, relational structure includes not only links between tables, but also functional dependencies among attributes within individual relations. Recent graph-based relational learning has begun to recognize the importance of such dependencies. For example, FROG explicitly regularizes learned graph representations to preserve functional-dependency constraints inherent in the underlying database (Huang et al., 2026). However, these dependencies are used primarily to constrain learned representations rather than to construct explicit tabular features. In the feature-compilation setting, it remains largely unexplored whether functional dependencies can instead guide which relational information should be compiled into explicit features.

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.

We take this view, using dependency structure as a feature-construction signal rather than only as a constraint on learned representations.

Problem: marginal aggregation cannot represent dependency structure. Conventional relational feature compilation does not directly capture this dependency structure. Methods such as DFS aggregate related records using statistics such as counts, means, cardinalities, and recency (Kanter & Veeramachaneni, 2015), which primarily summarize individual columns rather than relationships between them. As illustrated in Figure 1, two relational histories can therefore share the same aggregate representation while differing in their cross-column dependency structure $X \rightarrow A$. In three synthetic constructions, Canonical DFS remains unchanged while functional-dependency hypergraph (FDHG) features detect the dependency change. In real-data rewiring on `rel-stack/user-badge`, Canonical DFS remains unchanged for every accepted intervention, whereas FDHG changes for 43.96% of them (Appendix A).

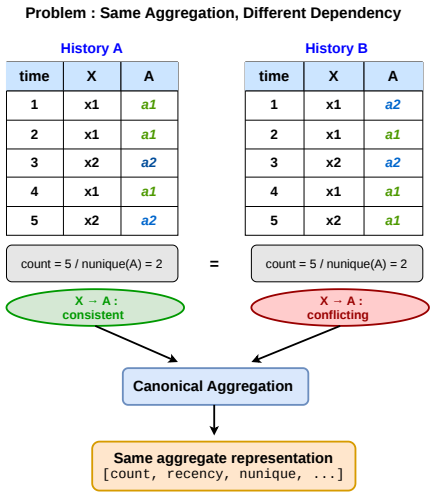

Figure 1: Canonical aggregation can collapse determinant-dependent structure, whereas FDHG exposes dependency information that remains unresolved by marginal aggregation. Our compiler expands the candidate representation with dependency-aware features and activates them only when supported by train-only temporal evidence.

Approach. We first construct a compact dependency-agnostic base representation by automatically generating marginal relational features such as counts, means, cardinalities, and recency features. We then use a functional-dependency hypergraph (FDHG) to add candidate features that capture cross-column relationships $X \rightarrow A$. For each candidate dependency $X \rightarrow A$, FDHG summarizes the dependency structure observed in the available history and compiles it into fixed-length point-in-time features. Because dependency-aware features are not uniformly useful, DARC adds them selectively. Instead, Independent selection evaluates each candidate individually against the base representation, while Greedy selection evaluates its conditional gain given the dependencies already selected. Finally, a hierarchical gate selects the final representation using only train-only temporal evidence.

Results. The hierarchical gate selects the dependency-agnostic base representation or an FD-enhanced variant of the base on 48 of the 51 tasks, including an FD-enhanced variant on 14 tasks. The hierarchical gate selects the dependency-agnostic base representation or an FD-enhanced variant of the base on 48 of the 51 tasks, including an FD-enhanced variant on 14 tasks. In budget-controlled comparisons, our selected representations achieve positive win–loss margins against JUICE ($30/1/7$) (Xu et al., 2026), FastProp ($20/1/15$) (getML, 2026), and matched Canonical DFS ($28/4/9$) (Kanter & Veeramachaneni, 2015). The benefits of the selected representation are observed across six downstream tabular predictors, indicating that they are not specific to the decoder used for train-only selection.

Our contributions are:

- We identify a limitation of conventional relational feature compilation: marginal aggregates can miss cross-column dependency structure.
- We introduce DARC, a dependency-aware relational compiler that converts functional-dependency structure into explicit point-in-time features and adaptively integrates them with conventional relational representations.
- We design Independent and Greedy selection strategies, together with a hierarchical gate, to activate dependency-aware features only when supported by train-only evidence.

- We show that Independent and Greedy selection, together with a hierarchical gate, activate dependency-aware features only where train-only evidence supports them, and that the resulting representation transfers across 6 downstream tabular predictors.

2 Related Work

2.1 Relational Prediction and Representation Learning

Relational prediction methods either learn directly from linked database structure or first compile relational history into tabular features. Relational Deep Learning formalizes graph-based learning over relational databases (Fey et al., 2024), while RelBench provides a benchmark suite for evaluating relational prediction methods (Robinson et al., 2024). Subsequent work explores learned relational representations through graph-centric inference and message passing (Wang et al., 2024; Chen et al., 2025), as well as relational foundation models and transfer (Wang et al., 2025; Ranjan et al., 2026; Fey et al., 2025; Hudovernik et al., 2026). Recent work such as FROG further incorporates functional-dependency constraints into learned relational representations (Huang et al., 2026). Our work is complementary: rather than constraining a learned relational encoder, we use dependency structure to determine which explicit relational features should be compiled for a downstream tabular predictor.

2.2 Automatic Relational Feature Engineering

Deep Feature Synthesis (DFS) constructs relational features by composing relational traversals with aggregation and transformation primitives (Kanter & Veeramachaneni, 2015). OneBM extends automatic feature engineering to relational databases through systematic feature extraction from related tables (Lam et al., 2017). ARDA studies automatic relational data augmentation for machine-learning pipelines (Chepurko et al., 2020), while GFS performs graph-based feature synthesis for prediction over relational databases (Zhang et al., 2024). More recently, JUICE, implemented in the RDBLearn system, constructs parameter-free column-preserving relational representations for downstream tabular foundation models (Xu et al., 2026). These approaches primarily summarize values along relational paths or within individual source columns. Our FDHG construction instead represents how consistently one source attribute determines another by compiling the empirical relationship $X \to A$ into an explicit point-in-time feature primitive.

2.3 Functional Dependencies and Dependency-Aware Data Profiling

Functional dependencies are traditionally studied as database constraints for schema design, dependency discovery, and data-quality analysis. TANE discovers exact and approximate functional dependencies using partition-based search (Huhtala et al., 1999), while HyFD combines sampling- and lattice-based techniques for functional-dependency discovery (Papenbrock & Naumann, 2016). Kruse and Naumann study efficient discovery of approximate dependencies (Kruse & Naumann, 2018), and Mandros et al. develop statistically reliable approximate-FD measures based on information-theoretic dependence (Mandros et al., 2017). More recent work studies FD discovery through hitting-set enumeration (Bleifuß et al., 2024). Our objective differs from dependency discovery itself. Classical dependency-discovery methods primarily aim to identify or score schema-level relationships such as $X \to A$. Within DARC, FDHG uses dependency structure as an explicit relational compilation primitive, converting local dependency ambiguity into point-in-time predictive features.

3 Preliminaries

3.1 Relational Prediction

A relational database $\mathcal{D} = \{T_1, \ldots, T_M\}$ contains tables connected by primary–foreign-key links; $\tau_r(u)$ denotes the timestamp of row u in source table T_r. A prediction instance $q_i = (e_i, t_i, y_i)$ specifies a target entity, prediction time, and label. For a reachable source table T_r, its point-in-time history is $H_{ir} = \{u \in T_r : u \text{ is linked to } e_i,\ \tau_r(u) \leq t_i\}$. Let ψ denote the compiler's fitted

quantities. A compiler maps $H_i = \{H_{ir}\}_r$ to fixed-length features $x_i = \Phi(H_i; \psi)$. A downstream tabular predictor with parameters θ outputs $\hat{y}_i = g_\theta(x_i)$.

3.2 Temporal Validity

Each history H_i contains only source records available by the prediction time t_i (Robinson et al., 2024). All fitted compiler quantities, including feature-selection and representation-selection decisions, use only the official training split. Section 4.1 describes the expanding temporal folds used to enforce this requirement.

3.3 Column-wise relational aggregation

Conventional compilers such as DFS (Kanter & Veeramachaneni, 2015) build x_i from aggregation declarations $\alpha = (r, c, o)$ consisting of a source table, a column, and an aggregation operator. Each declaration summarizes a single column of the history, $x_{i,\alpha} = o\big(\llbracket u.c : u \in H_{ir} \rrbracket\big)$ (or the row timestamps), and a declaration set $\mathcal{A}$ yields $x_i^{\mathcal{A}} = [x_{i,\alpha}]_{\alpha \in \mathcal{A}}$; Canonical DFS uses the fixed declaration set of its standard configuration (Appendix B.1). Because each feature is computed from one column at a time, such a representation is invariant to how columns co-vary: permuting the values of a column A across the rows of H_{ir} leaves $x_i^{\mathcal{A}}$ unchanged, yet can change how consistently another column X determines A (Figure 1). Conditional aggregates could recover part of this joint structure, but only after choosing which column pairs to condition on; we use functional-dependency structure to make that choice.

3.4 Functional Dependencies

Let X and A be attributes of source table T_r. For rows $R \subseteq T_r$, let $n_{x,a}(R) = |\{u \in R : u.X = x,\ u.A = a\}|$. The functional dependency $X \to A$ holds exactly on R if every observed x co-occurs with a single value of A, i.e., $|\{a : n_{x,a}(R) > 0\}| = 1$ (Silberschatz et al., 2019). In real data, we instead treat $X \to A$ as an *empirical dependency* characterized by $p_R(a \mid x) = \frac{n_{x,a}(R)}{\sum_{a'} n_{x,a'}(R)}$, whose strength can vary across determinant values: some x map almost uniquely to A, whereas others remain ambiguous.

4 Method

4.1 Design Goals and Overview

Our goal is to compile relational history into a compact fixed-length representation that preserves useful dependency structure without indiscriminate feature expansion or information leakage. DARC builds a dependency-agnostic *Base* from column-wise aggregates (Section 4.2) and augments it with point-in-time features for candidate dependencies $X \to A$ (Section 4.3). Independent and Greedy selection retain useful dependencies (Section 4.4); a hierarchical gate then chooses among Canonical DFS, Base, and the two FD-enhanced variants (Section 4.5).

Train-only evidence. All data-dependent decisions, including candidate discovery, feature fitting, and selection, use only the official training split. Its target instances form $F = 3$ expanding temporal folds $\{(\mathcal{I}_{\mathrm{tr}}^{(f)}, \mathcal{I}_{\mathrm{va}}^{(f)})\}_{f=1}^{F}$, with inner training preceding inner validation. A fixed decoder scores candidate representations on these folds; the selected representation is frozen before official validation and reused across downstream predictors.

4.2 Automatic Base Representation

Let $\mathcal{A}_{\mathrm{cand}}$ denote the task-specific candidate aggregation declarations. Using inner temporal training evidence, DARC selects a subset $\mathcal{A}^\star \subseteq \mathcal{A}_{\mathrm{cand}}$ under the configured feature budget and compiles the point-in-time Base representation $x_{\mathrm{Base},i} = x_i^{\mathcal{A}^\star}$. Candidate generation and budget selection are detailed in Appendices B.3 and B.4.

4.3 Dependency-Aware Feature Construction

Functional-dependency hypergraph. We organize candidate dependencies in a task-specific hypergraph $\mathcal{H}_{\mathrm{FD}} = (V, E)$: vertices are eligible source attributes, and each directed hyperedge represents an empirical dependency $d = (r, X \to A)$ between attributes of source table T_r. The notation permits multiple determinant attributes, although our implementation uses a single attribute X. Each edge yields an FDHG feature block.

Train-only candidate discovery. Using the earliest inner-training fold, we discover $\mathcal{E}_{\mathrm{cand}} = \mathrm{DiscoverFD}(\mathcal{D}, \mathcal{I}_{\mathrm{tr}}^{(1)})$ and set $E = \mathcal{E}_{\mathrm{cand}}$. Eligible pairs are filtered by support and ambiguity, ranked, and capped at $B_{\mathrm{cand}} = 32$. This candidate set is frozen before Independent or greedy selection.

Point-in-time feature compilation. For fold f and candidate $d = (r, X \to A)$, let $R_r^{(f)}$ be the rows of T_r observed by the end of fold f's inner-training window, pooled across all entities, and let $n_{x,a}^{(f,d)} = n_{x,a}(R_r^{(f)})$ (Section 3.4). FDHG refits the conditional map within each fold:

$$p^{(f,d)}(a \mid x) = \frac{n_{x,a}^{(f,d)}}{\sum_{a'} n_{x,a'}^{(f,d)}}.$$

For target instance q_i, let $u_{i,d}^{\star}$ be its most recent linked source row with $\tau_r(u_{i,d}^{\star}) \le t_i$, and set $x_{i,d} = X(u_{i,d}^{\star})$. FDHG queries $p^{(f,d)}(\cdot \mid x_{i,d})$; if no such row exists or $x_{i,d}$ was unseen in fold training, the block is marked missing. The target's own history thus contributes only $x_{i,d}$, while the distribution it indexes is estimated from all entities' rows (Figure 2).

Let $p_{(1)}^{(f,d)}(x)$ and $p_{(2)}^{(f,d)}(x)$ denote the largest and second-largest probabilities in $p^{(f,d)}(\cdot \mid x)$, respectively; set $p_{(2)}^{(f,d)}(x) = 0$ when only one outcome is observed. FDHG summarizes the queried dependency state using

$$\phi_{\mathrm{conf}}^{(f,d)}(x) = \max_a p^{(f,d)}(a \mid x), \qquad \phi_{\mathrm{ent}}^{(f,d)}(x) = -\sum_a p^{(f,d)}(a \mid x) \log p^{(f,d)}(a \mid x),$$

$$\phi_{\mathrm{conflict}}^{(f,d)}(x) = \left|\left\{a : n_{x,a}^{(f,d)} > 0\right\}\right|, \qquad \phi_{\mathrm{support}}^{(f,d)}(x) = \sum_a n_{x,a}^{(f,d)},$$

$$\phi_{\mathrm{margin}}^{(f,d)}(x) = p_{(1)}^{(f,d)}(x) - p_{(2)}^{(f,d)}(x).$$

ϕ_{conf} measures local dependency strength (Kivinen & Mannila, 1995); ϕ_{ent} measures conditional ambiguity (Mandros et al., 2017); ϕ_{conflict} summarizes the number of conflicting dependent outcomes (Huhtala et al., 1999); ϕ_{support} records the amount of evidence underlying the queried state (Mandros et al., 2017); and ϕ_{margin} measures the separation between the two most likely outcomes (Nguyen et al., 2022).

Each statistic has a missingness indicator; together they form the block $\widetilde{\boldsymbol{\phi}}_i^{(f,d)}$. Let $x_{\mathrm{Base},i}^{(f)}$ denote the Base fitted on fold f's inner-training data. For a selected dependency set $\mathcal{S}$, the augmented representation is

$$x_{\mathrm{Base+FDHG},i}^{(f)}(\mathcal{S}) = x_{\mathrm{Base},i}^{(f)} \,\|\, \Big\|_{d \in \mathcal{S}} \widetilde{\boldsymbol{\phi}}_i^{(f,d)}.$$

Entirely missing or constant columns are removed before model fitting.

4.4 Train-Only Dependency Selection

Let $s_f(z)$ be the inner-validation score of representation z on fold f, and let $\bar{s}(z) = F^{-1} \sum_{f=1}^{F} s_f(z)$. For $\mathcal{S} \subseteq \mathcal{E}_{\mathrm{cand}}$, $s_f(\mathcal{S})$ denotes the score of Base augmented with $\mathcal{S}$; thus $s_f(\varnothing)$ is the Base score. To make positive values indicate improvement for either metric direction, define

$$\mathrm{Imp}(s_{\mathrm{cand}}, s_{\mathrm{ref}}) = \begin{cases} s_{\mathrm{cand}} - s_{\mathrm{ref}}, & \text{higher is better,} \\ s_{\mathrm{ref}} - s_{\mathrm{cand}}, & \text{lower is better.} \end{cases}$$

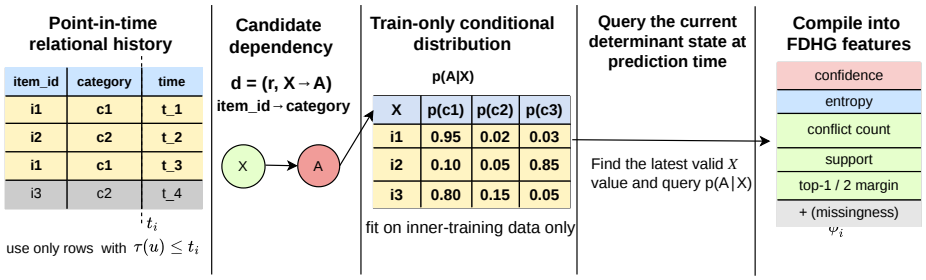

Figure 2: **Point-in-time compilation of one candidate dependency into FDHG features.** The compiler estimates $p(A \mid X)$ using fold-training history only, retrieves the most recent valid determinant value X available to each target by its prediction time, and converts the queried distribution into five structural statistics and missingness indicators.

For either selector, let $\Delta^{(f)}$ be a candidate's gain on fold f, and define its mean gain as $\overline{\Delta} = F^{-1} \sum_{f=1}^{F} \Delta^{(f)}$. A candidate passes screening if $\sum_{f=1}^{F} \mathbf{1}[\Delta^{(f)} > 0] \geq 2$ and $\overline{\Delta} > 0$.

Independent screening. Independent evaluates each dependency relative to Base,

$$\Delta_{\mathrm{Ind}}^{(f)}(d) = \mathrm{Imp}(s_f(\{d\}), s_f(\varnothing)),$$

and retains passing dependencies up to the selected-dependency budget B_{sel}.

Greedy selection. Given selected set $\mathcal{S}_j$, Greedy instead evaluates conditional gain,

$$\Delta_{j}^{(f)}(d) = \mathrm{Imp}(s_f(\mathcal{S}_j \cup \{d\}), s_f(\mathcal{S}_j)).$$

It repeatedly adds the passing dependency with the highest mean conditional gain until no candidate passes or the budget is reached. If no singleton passes initially, a train-only pairwise initialization may be used. Full screening and selection details are given in Appendices B.7, B.8, and B.12.

4.5 Hierarchical Representation Selection

Dependency-level selection does not determine whether an FD-enhanced representation should replace a simpler alternative. We therefore select the final representation from

$$\mathcal{V} = \{v_{\mathrm{DFS}}, v_{\mathrm{Base}}, v_{\mathrm{Ind}}, v_{\mathrm{Greedy}}\}$$

using the mean inner-fold score $\bar{s}(v)$.

Given a strategy-level tolerance $\varepsilon_{\mathrm{strat}}$, the gate first selects a baseline between Canonical DFS and the dependency-agnostic Base. DFS replaces Base only when

$$\mathrm{Imp}(\bar{s}(v_{\mathrm{DFS}}), \bar{s}(v_{\mathrm{Base}})) > \varepsilon_{\mathrm{strat}}.$$

Otherwise, Base is retained. Let v_B denote the selected baseline and $s_B = \bar{s}(v_B)$. An FD-enhanced strategy is admissible only when

$$\mathrm{Imp}(\bar{s}(v), s_B) > \varepsilon_{\mathrm{strat}}, \qquad v \in \{v_{\mathrm{Ind}}, v_{\mathrm{Greedy}}\},$$

If neither FD-enhanced strategy is admissible, the baseline is retained. If both are admissible, the better strategy is selected when their score difference exceeds $\varepsilon_{\mathrm{strat}}$; otherwise, the gate prefers the strategy with fewer selected dependencies, with Independent as the final deterministic tie-breaker. The resulting strategy $v^{\star}$ is frozen before official-validation evaluation. Exact tolerances and pseudocode are provided in Appendices B.9 and B.13.

Table 1: Representative task-level comparison of relational feature compilers under a common TabPFN evaluation protocol. Values are means over four seeds; mean ± standard deviation results are reported in Table 15. We show seven representative tasks with complete four-seed results for all five compilers, chosen to span six dataset families and both classification and regression settings. Aggregate comparisons use all available paired tasks rather than this illustrative subset. Best performance in each row is shown in bold. †The train-only gate selects an FD-enhanced representation on this task; on the other tasks it selects Base.

Dataset	Task	Metric	Matched DFS	JUICE	FastProp	FastDFS	DARC
rel-f1	driver-top3	AUROC ↑	0.7541	0.7814	0.8727	0.8051	**0.8731**†
rel-f1	driver-position	RMSE ↓	4.6329	4.0087	3.8000	4.1294	**3.6330**†
rel-arxiv	author-publication	RMSE ↓	1.0605	1.0132	1.0791	1.0590	**0.9609**
rel-hm	item-sales	RMSE ↓	0.5755	0.6375	0.5819	0.5694	**0.5116**
rel-ratebeer	brewer-dormant	AUROC ↑	0.7644	0.7891	0.7978	0.8083	**0.8148**†
rel-amazon	item-churn	AUROC ↑	0.7519	0.7308	0.7795	0.7441	**0.7880**
rel-trial	study-adverse	RMSE ↓	369.2789	362.0842	345.1714	**320.7027**	343.8705

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Tasks and evaluation protocol. We evaluate 51 relational prediction tasks from 12 datasets using official splits and metrics. For each instance, features use only source records available by its prediction time. We compare materialized representations under the same downstream predictor within each task to assess differences in feature construction. The Canonical DFS inventory and full downstream-decoder results appear in Appendices B.1 and E, respectively.

Train-only representation construction. DARC selects among four representations: Canonical DFS, Base, and the Independent and Greedy FD-enhanced variants. Candidate construction and representation selection use only the official training split, with the $F = 3$ expanding temporal folds defined in Section 4.1. HGB serves as the fixed decoder for this selection. The Base feature budget is chosen from $\{4, 8, 12, 16\}$, and $B_{\text{cand}} = B_{\text{sel}} = 32$. The hierarchical gate selects the final representation, which is frozen before official-validation evaluation. Further screening rules and selection tolerances appear in Appendices B.7 and B.9.

Relational feature-compiler baselines. We compare DARC with Canonical DFS, JUICE, FastProp, and FastDFS by evaluating their materialized representations under the same downstream predictor and task-wise protocol. For the budget-controlled baselines, we use the realized model-input dimensionality K_i of the frozen DARC representation as the task-specific feature budget. The corresponding matching procedures for Canonical DFS, JUICE, and FastProp are detailed in Appendix D.2. Related systems such as GFS (Zhang et al., 2024) and RDBLearn (Xu et al., 2026) report end-to-end scores that combine representation construction with specific prediction pipelines. We therefore exclude those reported scores from this controlled feature-compiler comparison. For win/tie/loss summaries, we compare the unrounded mean task metric; a tie denotes numerical equality at the stored precision, while otherwise the metric direction determines the win or loss.

Downstream predictor transfer. After the train-only gate freezes one representation per task, we evaluate it with TabPFN, XGBoost, CatBoost, HGB, TabICL, and RealMLP. We do not rerun representation construction or selection for each predictor; this tests whether choices made with HGB transfer to other tabular predictors. Where supported, multi-seed predictors use seeds 41–44, while HGB is evaluated deterministically.

5.2 COMPARISON WITH RELATIONAL FEATURE COMPILERS

Table 1 compares DARC with Canonical DFS, JUICE, FastProp, and FastDFS on representative tasks using the same TabPFN predictor and official task metrics. Matched Canonical DFS, JUICE, and FastProp use task-wise controlled feature budgets; FastDFS uses its materialized representation.

Across the seven representative tasks, DARC achieves the best mean performance on six, spanning both classification and regression and five dataset families. Relative to the strongest competing compiler, the margins are largest on `item-sales`, `author-publication`, and `driver-position`, whereas DARC and FastProp are nearly tied on `driver-top3` and FastDFS performs best on `study-adverse`. On four of these tasks the gate selects Base, so the gains there come from train-only Base construction; on the remaining three (†) it selects an FD-enhanced variant.

Across all tasks with paired evaluations, DARC achieves a win/tie/loss count of 28/4/9 against budget-matched Canonical DFS over 41 tasks, 30/1/7 against budget-matched JUICE over 38 tasks, and 20/1/15 against budget-matched FastProp over 36 tasks. Appendix D.1 reports full task-level results and coverage, Appendix D.2 feature counts, and Appendix D.3 a strict common-support comparison on the 32 tasks where all budget-matched compilers are available.

5.3 When Do Dependency-Aware Features Help?

DARC treats dependency-aware features as optional rather than universally useful. Its train-only gate chooses among three representation types: Canonical DFS, a dependency-agnostic base representation, and FD-enhanced variants of that base. Across the 51 tasks, the gate selects Canonical DFS on 3 tasks, the base representation on 34, and an FD-enhanced representation on 14 (Table 2). Thus, DARC selects the base or an FD-enhanced representation on 48 of 51 tasks, while using dependency-aware augmentation selectively when it provides additional value. The results show two sources of improvement. First, replacing Canonical DFS with the compact base representation accounts for most of the selected compiler representations. Second, dependency-aware augmentation adds smaller gains on a subset of tasks. Adding dependency features does not always improve performance. When the train-only evidence does not support them, DARC keeps the base representation.

Table 2: Representations selected by the train-only gate across 51 tasks.

Selected	Tasks	Fraction
Canonical DFS	3	5.9%
Base	34	66.7%
FD-enhanced	14	27.5%

Train-only selector diagnostics are consistent with this behavior. On the inner folds of four representative tasks, none of 80 cardinality-matched random dependency subsets scores above the Greedy-selected set, and adding all candidate dependencies lowers the inner-fold score on two of the four tasks (Appendix C.3). Across Base feature budgets, Greedy screening retains at least one dependency in 15 of 16 task–budget settings (Appendix C.1). Because these diagnostics reuse the inner folds that drive selection, they characterize the selection procedure rather than out-of-sample gains, which Tables 3 and 4 evaluate on the official validation split.

Table 3 illustrates the progression from Canonical DFS to Base and the FD-enhanced variants on representative tasks.

5.4 Transfer Across Downstream Predictors

DARC uses HGB as its selection decoder because representation selection requires repeatedly evaluating many candidate feature sets on train-only folds, for which a lightweight tabular predictor is practical. Importantly, HGB is the only decoder used to make representation-selection decisions. Once a representation is selected, it is frozen and is not re-selected for any downstream predictor.

We then test whether a representation selected with HGB remains useful under a different downstream model. We evaluate each frozen representation with HGB itself and with five alternative predictors: TabPFN, XGBoost, CatBoost, TabICL, and RealMLP.

Table 4 compares the frozen DARC representation with the dependency-agnostic base representation for each predictor. A win means that the frozen DARC-selected representation outperforms the dependency-agnostic base, while a loss means that the base performs better. Many ties are expected by construction: when DARC selects the base representation itself, the two representations are identical.

Table 3: Representative progression from Canonical DFS to the dependency-agnostic base and FD-enhanced variants. Values are mean performance over four seeds. Bold denotes the best observed mean in hindsight, whereas *Selected* reports the representation chosen beforehand by the train-only gate. *Independent yields no usable FD features at final refit on these tasks, so its final representation is identical to the base.

Task	Metric	Canonical DFS	Base	Independent	Greedy	Selected
driver-top3	AUROC ↑	0.7541	0.8361	0.8361*	**0.8731**	Greedy
driver-position	RMSE ↓	4.6329	3.6970	3.6970*	**3.6330**	Greedy
user-badge	AUROC ↑	0.8285	0.8428	0.8452	**0.8453**	Greedy
user-attendance	RMSE ↓	0.5228	0.4903	**0.4819**	0.4870	Greedy
user-engagement	AUROC ↑	0.8763	0.9027	0.9013	**0.9034**	Base
user-repeat	AUROC ↑	**0.7130**	0.6967	0.6807	0.6944	Canonical DFS

Table 4: Transfer of DARC's frozen representation across downstream predictors. Each entry reports the number of tasks on which DARC performs better / similarly / worse than the dependency-agnostic base representation. No representation is re-selected for a downstream predictor.

Downstream predictor	Tasks evaluated	Win / Tie / Loss	Win − Loss
TabPFN	50	11 / 34 / 5	+6
XGBoost	50	9 / 34 / 7	+2
CatBoost	48	13 / 33 / 2	+11
HGB	51	12 / 34 / 5	+7
TabICL	49	12 / 33 / 4	+8
RealMLP	51	9 / 35 / 7	+2

Across all six downstream predictors, DARC produces more wins than losses relative to the dependency-agnostic base. Because the representation is selected once using HGB and then frozen, without predictor-specific re-selection, these results indicate that the benefit of the selected representation is not confined to the decoder used during selection and transfers across heterogeneous tabular models.

6 Conclusion

We introduced DARC, a train-only adaptive compiler that augments conventional relational aggregates with point-in-time features derived from empirical functional dependencies.Independent and Greedy selection, together with a hierarchical gate, choose among conventional aggregation (Canonical DFS), the dependency-agnostic base, and FD-enhanced variants using only inner temporal evidence. Across 51 relational prediction tasks, the selected representations achieve favorable budget-controlled comparisons against relational feature-engineering baselines and transfer across heterogeneous downstream predictors. These results support dependency structure as a useful basis for adaptive relational feature construction, with functional-dependency hypergraphs providing a structured intermediate representation for candidate features. A current limitation of DARC is the computational cost of Greedy screening and its restriction to single-attribute determinants; extending the dependency search to richer determinant sets while retaining train-only, budgeted selection is an important direction for future work.

AI USE STATEMENT

Generative AI tools were used to assist with experimental-design feedback and with software implementation and debugging. They were also used to draft parts of some sections, to polish the writing, and to assist with information search. All AI-assisted methodological suggestions, drafted text, code, and interpretive content were reviewed and edited by the authors and verified against the underlying implementation, experimental artifacts, and cited sources. All reported experiments and numerical results were generated and validated using the authors' experimental pipeline. We take responsibility for the final content of this work, including text, claims, code, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies relational feature engineering using existing benchmark datasets and does not involve new human-subject data collection or user experimentation. We follow the access and evaluation protocols of the underlying benchmarks and do not release additional personally identifiable information. Any ethical considerations associated with individual prediction tasks therefore primarily inherit from the original datasets and benchmark settings.

REPRODUCIBILITY STATEMENT

We provide detailed descriptions of the proposed feature-construction and selection procedures in Section 4 and Appendix B, including candidate eligibility and ranking, point-in-time conditional fitting, feature budgets, screening criteria, strategy-gate tolerances, and deterministic tie-breaking. Appendix C reports selector ablations and efficiency measurements, Appendix D provides full task-level compiler comparisons, feature counts, selection diagnostics, and strict common-support analyses, and Appendix E reports complete task-level downstream-decoder results. Code, configuration files, and reproduction scripts for the main experiments are provided in the anonymous supplementary material.

REFERENCES

Tobias Bleifuß, Thorsten Papenbrock, Thomas Bläsius, Martin Schirneck, and Felix Naumann. Discovering functional dependencies through hitting set enumeration. *Proceedings of the ACM on Management of Data*, 2(1):1–24, 2024. doi: 10.1145/3639298.

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.

Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David R. Karger. Arda: Automatic relational data augmentation for machine learning. *Proceedings of the VLDB Endowment*, 13(9):1373–1387, 2020.

Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I. Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer. In *The Fourteenth International Conference on Learning Representations*, 2026.

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.

Matthias Fey, Vid Kocijan, Federico Lopez, Jan Eric Lenssen, and Jure Leskovec. Kumorfm: A foundation model for in-context learning on relational data. Technical report, Kumo AI, 2025.

getML. getml relbench: Relational feature engineering with fastprop. `https://github.com/getml/getml-relbench`, 2026. GitHub repository.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025. doi: 10.1038/s41586-024-08328-6.

Yi Huang, Qingyun Sun, Jia Li, Xingcheng Fu, and Jianxin Li. Is fixing schema graphs necessary? full-resolution graph structure learning for relational deep learning. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*. PMLR, 2026.

Valter Hudovernik, Federico López, Vid Kocijan, Akihiro Nitta, Jan Eric Lenssen, Jure Leskovec, and Matthias Fey. Kumorfm-2: Scaling foundation models for relational learning. *arXiv preprint arXiv:2604.12596*, 2026.

Ykä Huhtala, Juha Kärkkäinen, Pasi Porkka, and Hannu Toivonen. Tane: An efficient algorithm for discovering functional and approximate dependencies. *The Computer Journal*, 42(2):100–111, 1999.

James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics*, pp. 1–10. IEEE, 2015.

Jyrki Kivinen and Heikki Mannila. Approximate inference of functional dependencies from relations. *Theoretical Computer Science*, 149(1):129–149, 1995. doi: 10.1016/0304-3975(95)00028-U.

Sebastian Kruse and Felix Naumann. Efficient discovery of approximate dependencies. *Proceedings of the VLDB Endowment*, 11(7):759–772, 2018. doi: 10.14778/3192965.3192968.

Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327*, 2017.

Panagiotis Mandros, Mario Boley, and Jilles Vreeken. Discovering reliable approximate functional dependencies. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 355–363. ACM, 2017. doi: 10.1145/3097983.3098062.

Vu-Linh Nguyen, Mohammad Hossein Shaker, and Eyke Hüllermeier. How to measure uncertainty in uncertainty sampling for active learning. *Machine Learning*, 111:89–122, 2022. doi: 10.1007/s10994-021-06003-9.

Thorsten Papenbrock and Felix Naumann. A hybrid approach to functional dependency discovery. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 821–833. ACM, 2016. doi: 10.1145/2882903.2915203.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. Tabicl: A tabular foundation model for in-context learning on large data. In *ICML*, 2025.

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos I. Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational transformer: Toward zero-shot foundation models for relational data. In *The Fourteenth International Conference on Learning Representations*, 2026.

Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan Eric 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, 2024.

Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill Education, 7 edition, 2019.

Minjie Wang, Quan Gan, David Wipf, Zhenkun Cai, Ning Li, Jianheng Tang, Yanlin Zhang, Zizhao Zhang, Zunyao Mao, Yakun Song, Yanbo Wang, Jiahang Li, Han Zhang, Guang Yang, Xiao Qin, Chuan Lei, Muhan Zhang, Weinan Zhang, Christos Faloutsos, and Zheng Zhang. 4dbinfer: A 4d benchmarking toolbox for graph-centric predictive modeling on rdbs. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

Yanbo Wang, Xiyuan Wang, Quan Gan, Minjie Wang, Qibin Yang, David Wipf, and Muhan Zhang. Griffin: Towards a graph-centric relational database foundation model. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 64604–64627. PMLR, 2025.

Linjie Xu, Yanlin Zhang, Quan Gan, Minjie Wang, and David Wipf. No need to train your rdb foundation model. *arXiv preprint arXiv:2602.13697*, 2026.

Han Zhang, Quan Gan, David Wipf, and Weinan Zhang. Gfs: Graph-based feature synthesis for prediction over relational database. In *VLDB 2024 Workshop on Tabular Data Analysis*, 2024.

A Additional Motivation Experiments

A.1 Controlled Synthetic Aggregation Collisions

We construct paired relational histories whose determinant-dependent structures differ while their canonical DFS representations remain identical. Across all three synthetic families, the canonical DFS representation distance is exactly zero, whereas the FDHG representation distance is strictly positive.

Table 5: **Controlled aggregation-collision results.** Every paired construction has zero canonical DFS distance but positive FDHG distance.

Family	Canonical DFS distance	FDHG distance
A	0.000000	1.077319
B	0.000000	2.184185
C	0.000000	1.100260
Mean	0.000000	1.453921

These controlled collisions isolate a representational distinction: determinant-dependent structure can change even when the conventional relational aggregate vector remains fixed. The experiment measures representational sensitivity rather than predictive performance.

A.2 Real-Data DFS-Preserving Dependency Rewiring

We complement the synthetic constructions with a real-data intervention on the RelBench `rel-stack/user-badge` task. We consider the candidate dependency

$$\texttt{badges.Class} \rightarrow \texttt{badges.Name}.$$

The analysis unit is a target entity-time row. For each target row, only source records available no later than the target prediction timestamp are included. The intervention is therefore applied exclusively to train-only, point-in-time histories.

For each eligible history, we construct counterfactual versions by reassigning the observed `Name` values across source events while preserving their multiset. This changes the coupling between `Class` and `Name` without changing the number of events, determinant values, dependent-value marginal distribution, source timestamps, or missingness pattern.

The original and rewired histories are passed through the same canonical DFS and FDHG materialization procedures. We directly verify that the complete canonical DFS feature vector, including its missingness indicators, remains unchanged for every accepted intervention.

Table 6: **Audit of the real-data DFS-preserving rewiring experiment.** Rewiring is applied only to train-only, point-in-time histories.

Audit item	Value
Eligible target histories	5,000
Successfully rewired histories	4,977
Infeasible histories	23
Accepted interventions	23,556
Ineligible or infeasible attempts	98,986
Exact canonical DFS preservation	100.0%
Interventions with nonzero FDHG distance	43.9633%
Mean FDHG distance	0.130179
Median FDHG distance	0.000000

The corresponding FDHG edge is `badges.Class` → `badges.Name`. Prediction labels, official validation rows, and test rows are not accessed.

Across 23,556 accepted interventions, canonical DFS is preserved exactly in every case, yet 43.9633% of interventions induce a nonzero FDHG representation. Under ambiguity-increasing

rewiring, majority confidence and top-two margin decrease, whereas conditional entropy and conflict count increase. These results show that determinant-dependent coupling can vary while the complete canonical aggregate representation remains unchanged.

Preservation and leakage checks. For every accepted intervention, we additionally verify preservation of the target entity, target prediction timestamp, number of source events, determinant values, dependent-value multiset, source timestamps, missingness pattern, numeric summaries, cardinality features, recency features, and canonical DFS missing indicators. Prediction labels are not used, and no official validation or test rows are accessed. The observed FDHG differences therefore reflect sensitivity to determinant-dependent coupling rather than changes in conventional aggregate information.

B Additional Method Details

This appendix provides implementation details omitted from the main Method section for conciseness. All procedures follow the same train-only temporal protocol used in the main experiments.

B.1 Canonical DFS Construction Details

Canonical DFS is constructed using the task-specific relational paths and aggregation primitives defined by the canonical baseline configuration. Declarations are materialized under the same point-in-time constraint used throughout the paper. Depending on the source-column type and task schema, the canonical representation includes supported aggregation primitives such as counts, means, standard deviations, extrema, cardinalities, and recency summaries, together with their associated missingness indicators.

Table 7 reports the complete 51-task DFS inventory, including the number of full Canonical DFS declarations and realized model-input columns, the task-wise feature budget K_i used for the budget-controlled comparison, and the resulting matched-DFS declaration and column counts.

B.2 Budget-controlled Canonical DFS Comparison

For the budget-controlled comparison in Table 1, Canonical DFS is constructed separately from the Canonical DFS representation considered by the joint strategy gate.

For each task, let K_i denote the realized number of downstream model-input columns in our frozen selected representation. We use K_i as the task-specific DFS feature cap. Because DFS declarations are atomic and may emit multiple model-input columns, this protocol controls the available feature budget rather than enforcing exact equality in realized dimensionality. DFS declarations, including their associated missingness indicators, are therefore never split across the budget boundary.

If the complete Canonical DFS representation contains at most K_i model columns, it is retained in full. Otherwise, complete DFS declarations are selected using only the same three inner temporal training folds and the fixed HGB selection decoder used elsewhere in the train-only protocol. Official-validation metrics and test rows are not used for this selection.

When no complete DFS declaration fits within K_i, the smallest complete declaration is retained rather than splitting a logical DFS feature block. This produces a minimal one-column budget overflow on the affected tasks.

Budget-capped DFS representations are successfully constructed for all 51 tasks. For the TabPFN comparison, we require all four predefined evaluation seeds (41-44) to complete for both representations. Nine tasks are unavailable because the sampled matched-DFS representation contains no varying model input feature in at least one seed. One additional task, `rel-salt/sales-group`, is unsupported by TabPFN because its number of classes exceeds the supported limit. The resulting paired comparison therefore contains 41 tasks.

Because this budget-capped DFS baseline is constructed separately under the task-wise feature-cap protocol, it is not necessarily identical to the Canonical DFS representation considered by the joint strategy gate.

Table 7: **Canonical and budget-controlled DFS inventory across all 51 tasks.** Canonical columns denote the realized downstream model-input columns from the full Canonical DFS configuration. For the budget-controlled comparison, K_i is the realized feature count of our frozen selected representation. DFS declarations are treated as atomic blocks, including associated missingness indicators. A positive excess indicates that the smallest complete declaration was retained when no atomic declaration fit within K_i.

Dataset	Task	Full decl.	Full cols.	Ours K_i	Matched decl.	Matched cols.	Overflow
dbinfer-avs	repeater	7	14	4	1	2	–
dbinfer-retailrocket	cvr	3	6	4	1	2	–
rel-amazon	item-churn	7	14	4	2	4	–
rel-amazon	item-ltv	7	14	4	2	4	–
rel-amazon	review-rating	2	4	1	1	2	+1
rel-amazon	user-churn	7	14	4	2	4	–
rel-amazon	user-ltv	7	14	4	2	4	–
rel-arxiv	author-publication	2	4	2	1	2	–
rel-arxiv	paper-citation	2	4	6	2	4	–
rel-avito	ad-ctr	2	4	1	1	2	+1
rel-avito	searchinfo-isuserloggedon	6	12	4	2	4	–
rel-avito	searchstream-click	6	12	8	3	6	–
rel-avito	user-clicks	7	14	5	2	4	–
rel-avito	user-visits	2	4	1	1	2	+1
rel-event	event_interest-interested	6	12	13	6	12	–
rel-event	event_interest-not_interested	6	12	15	6	12	–
rel-event	user-attendance	6	12	30	6	12	–
rel-event	user-ignore	7	14	31	7	14	–
rel-event	user-repeat	7	14	14	7	14	–
rel-event	users-birthyear	6	12	1	1	2	+1
rel-f1	driver-dnf	6	12	27	6	12	–
rel-f1	driver-position	2	4	35	2	4	–
rel-f1	driver-top3	6	12	35	6	12	–
rel-f1	qualifying-position	6	12	5	2	4	–
rel-f1	results-position	2	4	6	2	4	–
rel-hm	item-sales	6	12	7	3	6	–
rel-hm	transactions-price	2	4	2	1	2	–
rel-hm	user-churn	6	12	4	2	4	–
rel-ratebeer	beer-churn	7	14	8	4	8	–
rel-ratebeer	beer_ratings-total_score	3	6	17	3	6	–
rel-ratebeer	brewer-dormant	7	14	24	7	14	–
rel-ratebeer	user-churn	7	14	27	7	14	–
rel-ratebeer	user-count	7	14	7	3	6	–
rel-salt	item-incoterms	7	14	2	1	2	–
rel-salt	item-plant	7	14	2	1	2	–
rel-salt	item-shippoint	7	14	2	1	2	–
rel-salt	sales-group	7	14	1	1	2	+1
rel-salt	sales-incoterms	7	14	14	7	14	–
rel-salt	sales-office	7	14	14	7	14	–
rel-salt	sales-payterms	7	14	2	1	2	–
rel-salt	sales-shipcond	7	14	1	1	2	+1
rel-stack	post-votes	2	4	14	2	4	–
rel-stack	user-badge	3	6	37	3	6	–
rel-stack	user-engagement	3	6	7	3	6	–
rel-trial	eligibilities-adult	2	4	5	2	4	–
rel-trial	eligibilities-child	2	4	14	2	4	–
rel-trial	site-success	2	4	1	1	2	+1
rel-trial	studies-enrollment	3	6	4	1	2	–
rel-trial	studies-has_dmc	2	4	67	2	4	–
rel-trial	study-adverse	3	6	4	2	4	–
rel-trial	study-outcome	2	4	56	2	4	–

B.3 Base Feature Construction Details

The base representation is constructed from schema- and task-derived candidate feature declarations. Each declaration is represented as

$$\alpha = (r, c, o),$$

where r denotes a source relation, c a source column, and o a point-in-time aggregation operator.

For each task, the compiler first resolves the target entity, prediction timestamp, relational paths, and eligible historical source relations. It then excludes columns that are unsafe or unsuitable for predictive aggregation, including target columns, identifiers, key columns, timestamps used only for temporal alignment, and unsupported high-cardinality or text-like attributes. From the remain-

ing source columns, the base representation generates candidate declarations using the aggregation operators supported by the corresponding column type and relation.

Each candidate declaration is materialized using only source rows available no later than the target prediction timestamp. Candidate utility is evaluated exclusively on the inner temporal training folds constructed within the official training split. Feature declarations are ranked and selected deterministically under the configured base feature budget, and the selected declaration set is frozen before official-validation evaluation.

When train-only budget selection is enabled, the compiler evaluates the budget grid described in Section B.4. Among budgets whose mean inner-fold score lies within the predefined tolerance of the best observed score, the smallest budget is retained. The resulting declarations are then rematerialized independently within each temporal fold and again for the final frozen representation.

The implementation records the selected declaration identities, source relations, source columns, aggregation operators, resolved feature budget, usable model-column count, temporal-fold scores, and strategy-selection provenance. This allows the base representation used by the joint gate to be reconstructed independently of the downstream decoder.

B.4 Base Feature Budget Selection

The base feature-construction procedure supports fixed, schema-scaled, and train-only grid budget policies. When train-only grid selection is enabled, the candidate budgets are

$$\mathcal{B} = \{4, 8, 12, 16\}.$$

Let $s(b)$ denote the mean inner-fold primary-metric score obtained using base feature budget b. The best observed score is

$$s^\star = \begin{cases} \max_{b \in \mathcal{B}} s(b), & \text{for higher-is-better metrics,} \\ \min_{b \in \mathcal{B}} s(b), & \text{for lower-is-better metrics.} \end{cases}$$

The budget-selection tolerance is

$$\tau_{\text{budget}} = \begin{cases} 0.001, & \text{for classification metrics,} \\ 0.001\,|s^\star|, & \text{for regression metrics.} \end{cases}$$

Among budgets whose score lies within τ_{budget} of the best observed score, the smallest budget is selected. This favors a more compact base representation when additional features do not provide a meaningful train-only improvement.

This procedure is used only when train-only grid selection is enabled; otherwise, the configured fixed or schema-scaled budget is used directly.

B.5 Candidate-Pool Freezing, Dependency Budgets, and Feature Usability

Candidate column eligibility. Eligible categorical and low-cardinality numeric attributes are capped at cardinality 256, and columns whose non-null unique ratio exceeds 0.95 are excluded. Text-like columns are additionally excluded if their mean string length exceeds 80 characters, maximum length exceeds 256 characters, or mean token count exceeds 12. Columns with long-text-like names are excluded when their cardinality exceeds 16. The minimum determinant and edge support is 2, and the minimum source coverage is strictly greater than 0.1.

The FD candidate space is discovered once using only the earliest inner-training fold:

$$\mathcal{E}_{\text{cand}} = \text{DiscoverFD}\left(\mathcal{D}, \mathcal{I}_{\text{tr}}^{(1)}\right).$$

The ordered candidate pool is then frozen before either selector is run. Independent and Greedy therefore operate on the identical search space:

Table 8: Sensitivity to the FDHG candidate-pool budget. Scores are train-only mean inner-fold scores; arrows indicate metric direction. The selected-edge budget is fixed at $B_{\text{sel}} = 32$.

Task	Metric	$B_{\text{cand}} = 16$	$B_{\text{cand}} = 32$	$B_{\text{cand}} = 64$
study-adverse	RMSE ↓	373.8609	373.6357	373.6357
driver-position	RMSE ↓	5.0861	5.0851	5.0851
studies-enrollment	RMSE ↓	107173.5	107173.5	**106870.6**
study-outcome	AUROC ↑	0.5735	0.5796	**0.5835**

$$\mathcal{E}_{\text{cand}}^{\text{Ind}} = \mathcal{E}_{\text{cand}}^{\text{Greedy}} = \mathcal{E}_{\text{cand}}.$$

The implementation computes a SHA-256 digest of the exported candidate file and verifies that both selector runs use the same candidate pool. We distinguish a candidate-edge budget B_{cand}, which bounds the number of discovered dependencies admitted to screening, and a selected-edge budget B_{sel}, which bounds the number of dependency edges retained in the final augmentation.

Each selected dependency d declares five structural statistics and one missingness indicator for each statistic, yielding at most ten columns before fold-specific usability filtering. Columns that are entirely missing or have zero variance within a fold are removed before decoder fitting. Thus,

$$0 \le u(f, d) \le 10,$$

where $u(f, d)$ is the number of usable FDHG columns contributed by dependency d in fold f. For a selected dependency set $\mathcal{S}$, the FDHG augmentation therefore contributes at most

$$\dim\left(x^{(f)}_{\text{Base+FDHG},i}(\mathcal{S})\right) - \dim\left(x^{(f)}_{\text{Base},i}\right) = \sum_{d \in \mathcal{S}} u(f, d) \le 10|\mathcal{S}|.$$

Base and dependency-derived feature blocks are aligned using deterministic target keys containing the target entity identifier and prediction timestamp. Duplicate target keys are rejected, and copied target labels are checked for consistency.

Candidate ranking and truncation. For each eligible candidate $X \to A$, let

$$\rho_{\text{conflict}} = \frac{|\{x : |\{a : n_{x,a} > 0\}| > 1\}|}{|\{x : \sum_a n_{x,a} > 0\}|}$$

denote the fraction of observed determinant values exhibiting conflicting dependent outcomes, and let

$$N_{\text{support}} = \sum_{x,a} n_{x,a}$$

denote its non-missing pair support. Candidates are ordered by decreasing ρ_{conflict}, then decreasing N_{support}, with remaining ties broken lexicographically by source table, determinant attribute, and dependent attribute. The ordered list is then truncated to the first B_{cand} candidates, after which edge ranks are assigned.

B.6 Sensitivity to the Candidate-Pool Budget

We vary the maximum FDHG candidate-pool size over $B_{\text{cand}} \in \{16, 32, 64\}$ on four representative tasks while fixing $B_{\text{sel}} = 32$ and otherwise retaining the same train-only selection protocol. Increasing the cap from 16 to 32 improves or preserves the inner-fold score on all four tasks. Expanding beyond 32 leaves two tasks unchanged while yielding additional improvements on two tasks, indicating that candidate utility does not universally saturate at 32.

For tasks whose eligible candidate pool exceeds 32, the larger search space also increases the cost of repeated Greedy evaluation. We therefore use $B_{\text{cand}} = 32$ as a bounded computational search budget that trades off candidate coverage against selection cost, rather than assuming that all useful dependencies occur within the first 32 candidates.

B.7 Independent Edge Screening Details

Independent screening evaluates each candidate dependency separately relative to the base representation. For candidate dependency $d \in \mathcal{E}_{\text{cand}}$ and inner temporal fold f,

$$\Delta_{\text{Ind}}^{(f)}(d) = \text{Imp}\left(s_f(\{d\}), s_f(\varnothing)\right).$$

The number of folds exceeding the edge-level gain threshold is

$$C_{\text{Ind}}(d) = \sum_{f=1}^{F} \mathbf{1}\left[\Delta_{\text{Ind}}^{(f)}(d) > \delta_{\text{dep}}\right].$$

Under the default joint-pipeline configuration, the screening rule is `fixed_count` with three temporal folds, minimum fold-level gain zero, and at least two passing folds. Hence,

$$F = 3, \qquad k = 2, \qquad \delta_{\text{dep}} = 0.$$

The comparison is strict: a fold passes only when its gain is strictly positive. A dependency is retained only when

$$C_{\text{Ind}}(d) = \sum_{f=1}^{3} \mathbf{1}\left[\Delta_{\text{Ind}}^{(f)}(d) > 0\right] \geq 2$$

and its mean gain is also strictly positive,

$$\overline{\Delta}_{\text{Ind}}(d) = \frac{1}{3}\sum_{f=1}^{3} \Delta_{\text{Ind}}^{(f)}(d) > 0.$$

Passing dependencies are ranked deterministically and truncated to the selected-dependency budget B_{sel}.

B.8 Greedy Pairwise Initialization

Greedy selection starts from

$$\mathcal{S}_0 = \varnothing$$

and first searches for an admissible single-dependency initialization. If no single dependency passes the forward screening rule, the implementation may evaluate candidate pairs

$$\{d_a, d_b\} \subseteq \mathcal{E}_{\text{cand}}.$$

The fold-level pair gain is

$$\Delta_{\text{pair}}^{(f)}(d_a, d_b) = \text{Imp}\left(s_f(\{d_a, d_b\}), s_f(\varnothing)\right).$$

A pair is admissible only when its fold-level and aggregate gains satisfy the same forward screening rule used by ordinary Greedy selection. Pairwise initialization therefore does not bypass the train-only selection criterion. If at least one pair passes, the highest-gain passing pair initializes the selected set. Greedy selection then continues by evaluating the conditional marginal utility of each remaining dependency.

B.9 Strategy-Gate Tolerance and Tie Handling

The joint gate applies a task-specific tolerance $\varepsilon_{\text{strat}}$ to baseline selection, FDHG admissibility, and comparison between admissible FDHG strategies. Under the default joint-gate configuration,

$$\varepsilon_{\text{strat}} = \begin{cases} 0.001, & \text{for classification metrics,} \\ 0.001\,|\bar{s}(v_{\text{Base}})|, & \text{for regression metrics.} \end{cases}$$

Canonical DFS replaces Base only when

$$\text{Imp}\left(\bar{s}(v_{\text{DFS}}), \bar{s}(v_{\text{Base}})\right) > \varepsilon_{\text{strat}}.$$

Otherwise, Base is retained.

Let $v_B \in \{v_{\text{DFS}}, v_{\text{Base}}\}$ denote the baseline selected by this comparison, and let

$$s_B = \bar{s}(v_B)$$

denote its mean train-only score. An FDHG strategy $v \in \{v_{\text{Ind}}, v_{\text{Greedy}}\}$ is admissible only when

$$\text{Imp}\left(\bar{s}(v), s_B\right) > \varepsilon_{\text{strat}}.$$

When both FDHG strategies are admissible, define

$$g_{\text{G}\leftarrow\text{I}} = \text{Imp}\left(\bar{s}(v_{\text{Greedy}}), \bar{s}(v_{\text{Ind}})\right).$$

If

$$g_{\text{G}\leftarrow\text{I}} > \varepsilon_{\text{strat}},$$

Greedy is selected. If

$$-g_{\text{G}\leftarrow\text{I}} > \varepsilon_{\text{strat}},$$

Independent is selected. Otherwise, the two strategies are treated as equivalent under the strategy tolerance and the strategy with fewer selected dependencies is preferred. If their selected dependency counts are also equal, Independent is used as the deterministic fallback.

The implementation additionally records whether the two FDHG scores are numerically equal within

$$\tau_{\text{exact}} = 10^{-12}.$$

This exact-tie quantity is provenance metadata only and does not alter the selection rule.

B.10 Safety, Selected Artifacts, and Provenance

After the joint gate selects the final strategy $v^\star$, the corresponding representation is frozen for downstream evaluation. If Independent or Greedy is selected, the associated selected dependency set is retained. If Canonical DFS or the base is selected, the selected dependency set is empty.

For candidate dependency

$$d = (r, X \to A),$$

every target-side dependency lookup is required to satisfy

$$\tau_r\big(u^\star_{i,d}\big) \le t_i,$$

where $u^\star_{i,d}$ denotes the source row returned for target instance q_i and dependency d, and $\tau_r(\cdot)$ is the timestamp function for source relation T_r. The implementation raises an assertion error if a future source row is returned.

The pipeline records candidate and selected dependency counts, candidate-pool hashes, Independent and Greedy dependency identifiers, screening thresholds, usable feature counts, selected strategy, selection reason, official-validation usage flags, and test-access flags.

The official validation metric is not used by base feature selection, FDHG dependency selection, or the joint strategy gate. Component runs may materialize official-validation predictions for reporting or artifact completeness, but these values do not influence feature or strategy selection.

The test split is not accessed during feature construction, dependency selection, or joint strategy selection. These audits establish temporal validity, candidate-pool consistency, feature usability, and selection provenance. They do not themselves establish predictive benefit.

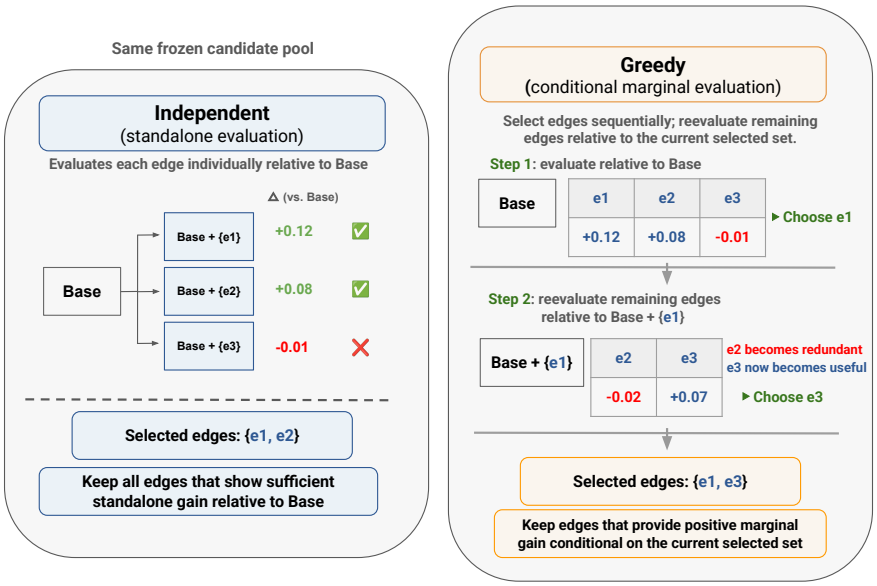

Figure 3: **Independent and Greedy selection evaluate dependency utility differently.** Independent screening measures the standalone gain of each candidate dependency when added to the base representation. Greedy selection instead measures the candidate's additional gain given the dependency blocks already selected, allowing utility to depend on the current representation context.

B.11 Independent and Greedy Selection Illustration

Independent and Greedy differ in whether candidate utility is evaluated standalone or conditionally on the current selected dependency set. Figure 3 illustrates this distinction.

B.12 Greedy Conditional Selection Algorithm

Algorithm 1 gives the complete Greedy conditional selection procedure used in the main experiments.

B.13 Full Hierarchical Strategy-Gate Algorithm

Algorithm 2 gives the full deterministic strategy-gate procedure summarized in Section 4.5.

Algorithm 1 Greedy Conditional FDHG Selection

Require: Frozen candidate pool $\mathcal{E}_{\text{cand}}$; inner temporal folds; selected-dependency budget B_{sel}; forward screening rule G
Ensure: Greedy-selected dependency set $\mathcal{E}_{\text{Greedy}}$

$\mathcal{S} \leftarrow \varnothing$
Evaluate each $d \in \mathcal{E}_{\text{cand}}$ relative to the base representation using

$$\text{Imp}\left(s_f(\{d\}), s_f(\varnothing)\right)$$

across the inner folds
Let $\mathcal{T}$ be the single candidates that pass G
if $\mathcal{T} \neq \varnothing$ **then**
    Add the highest-gain candidate in $\mathcal{T}$ to $\mathcal{S}$
else if pairwise initialization is enabled **then**
    Evaluate candidate pairs $\{d_a, d_b\} \subseteq \mathcal{E}_{\text{cand}}$ relative to the base representation using

$$\text{Imp}\left(s_f(\{d_a, d_b\}), s_f(\varnothing)\right)$$

    across the inner folds
    Retain only pairs that pass G
    if at least one pair passes **then**
        Initialize $\mathcal{S}$ with the highest-gain passing pair
    end if
end if
while $|\mathcal{S}| < B_{\text{sel}}$ **do**
    $\mathcal{T} \leftarrow \varnothing$
    for all $d \in \mathcal{E}_{\text{cand}} \setminus \mathcal{S}$ **do**
        Evaluate the conditional gain of d using

$$\text{Imp}\left(s_f(\mathcal{S} \cup \{d\}), s_f(\mathcal{S})\right)$$

    across the inner folds
        if d passes G **then**
            Add d and its aggregate conditional gain to $\mathcal{T}$
        end if
    end for
    if $\mathcal{T} = \varnothing$ **then**
        break
    end if
    Add the passing candidate with the largest aggregate conditional gain to $\mathcal{S}$
    Break ties by deterministic candidate order
end while
$\mathcal{E}_{\text{Greedy}} \leftarrow \mathcal{S}$
return $\mathcal{E}_{\text{Greedy}}$

Algorithm 2 Hierarchical Train-Only Strategy Gate

Require: Mean train-only strategy scores $\bar{s}(v_{\mathrm{DFS}})$, $\bar{s}(v_{\mathrm{Base}})$, $\bar{s}(v_{\mathrm{Ind}})$, $\bar{s}(v_{\mathrm{Greedy}})$; strategy tolerance $\varepsilon_{\mathrm{strat}}$; selected dependency counts J_{Ind} and J_{Greedy}
Ensure: Final selected strategy $v^{\star}$

if $\mathrm{Imp}\left(\bar{s}(v_{\mathrm{DFS}}), \bar{s}(v_{\mathrm{Base}})\right) > \varepsilon_{\mathrm{strat}}$ **then**
    $v_B \leftarrow v_{\mathrm{DFS}}$
else
    $v_B \leftarrow v_{\mathrm{Base}}$
end if
$s_B \leftarrow \bar{s}(v_B)$

$$\mathcal{V}_{\mathrm{adm}} \leftarrow \{v \in \{v_{\mathrm{Ind}}, v_{\mathrm{Greedy}}\} : \mathrm{Imp}\left(\bar{s}(v), s_B\right) > \varepsilon_{\mathrm{strat}}\}$$

if $\mathcal{V}_{\mathrm{adm}} = \varnothing$ **then**
    $v^{\star} \leftarrow v_B$
else if $|\mathcal{V}_{\mathrm{adm}}| = 1$ **then**
    $v^{\star} \leftarrow$ the sole strategy in $\mathcal{V}_{\mathrm{adm}}$
else
    $g_{\mathrm{G}\leftarrow\mathrm{I}} \leftarrow \mathrm{Imp}\left(\bar{s}(v_{\mathrm{Greedy}}), \bar{s}(v_{\mathrm{Ind}})\right)$
    if $g_{\mathrm{G}\leftarrow\mathrm{I}} > \varepsilon_{\mathrm{strat}}$ **then**
        $v^{\star} \leftarrow v_{\mathrm{Greedy}}$
    else if $-g_{\mathrm{G}\leftarrow\mathrm{I}} > \varepsilon_{\mathrm{strat}}$ **then**
        $v^{\star} \leftarrow v_{\mathrm{Ind}}$
    else if $J_{\mathrm{Greedy}} < J_{\mathrm{Ind}}$ **then**
        $v^{\star} \leftarrow v_{\mathrm{Greedy}}$
    else if $J_{\mathrm{Ind}} < J_{\mathrm{Greedy}}$ **then**
        $v^{\star} \leftarrow v_{\mathrm{Ind}}$
    else
        $v^{\star} \leftarrow v_{\mathrm{Ind}}$
    end if
end if
return $v^{\star}$

Table 9: **Sensitivity of Greedy FDHG to the cross-fold consistency requirement.** Gain is computed relative to the base representation using the task-specific metric direction, such that larger values always indicate improvement. The two-of-three setting is the default used throughout the main experiments.

Task	Positive folds	Selected dependencies	Inner score	Gain vs. Base
studies-enrollment	1/3	2	107180.31	+20.14
	2/3 (default)	3	**107179.84**	**+20.61**
	3/3	0	107200.45[a]	+0.00
study-outcome	1/3	7	**0.5981**	**+0.0161**
	2/3 (default)	6	0.5979	+0.0159
	3/3	3	0.5916	+0.0096

[a] Under the three-of-three criterion on `studies-enrollment`, no FDHG dependency passes screening. The strategy therefore falls back to the base representation; the displayed inner score is the base score rather than an FDHG score.

The one-of-three and three-of-three runs modify only the minimum number of positive inner folds. The candidate pool, base representation, temporal folds, metric, dependency budget, and minimum gain threshold are held fixed.

Official validation performance is not used to choose the default cross-fold requirement. All comparisons use train-only inner temporal evidence.

C Additional Selector Ablations

Sensitivity to cross-fold consistency. The default Greedy screening rule requires a dependency to obtain a strictly positive gain in at least two of the three inner temporal folds. To evaluate the sensitivity of this choice, we vary the required number of positive folds from one to three while keeping the minimum gain threshold fixed at zero. The one-of-three setting is more permissive, the two-of-three setting is our default, and the three-of-three setting requires improvement in every temporal fold.

Table 9 shows that the default two-of-three rule provides an intermediate operating point between permissive and overly strict screening. On `studies-enrollment`, the one-of-three rule selects two dependencies and improves RMSE over the base by 20.14, whereas the default two-of-three rule selects three dependencies and achieves a slightly larger improvement of 20.61. Requiring improvement in all three folds rejects every candidate dependency, causing the strategy to fall back to the base.

On `study-outcome`, the permissive one-of-three criterion achieves the largest numerical inner-fold AUROC, 0.5981, using seven selected dependencies. The default two-of-three rule achieves nearly the same score, 0.5979, while retaining six dependencies; the difference is only 0.0002 AUROC. In contrast, requiring improvement in all three folds reduces the selected set to three dependencies and lowers the inner-fold AUROC to 0.5916.

These results indicate that requiring evidence across multiple temporal folds filters fold-specific gains without requiring every useful dependency to improve in every fold. The default two-of-three criterion therefore provides a conservative balance between predictive utility and cross-fold consistency.

Pairwise initialization. Singleton screening can miss complementary dependencies for which neither dependency is sufficiently useful on its own, but the pair provides an admissible starting point. To address this case, Greedy includes a pairwise initialization mechanism that is invoked only when no singleton candidate passes the train-only screening criterion.

This mechanism is exercised on `rel-f1/driver-position`. Without pairwise initialization, all 32 singleton candidates fail screening, so Greedy cannot initialize and falls back to the preserved base representation, whose inner RMSE is 5.0883.

Table 10: Effect of pairwise initialization on `rel-f1/driver-position`. Scores are train-only mean inner temporal-fold RMSE; lower is better.

Setting	Selected FDHG dependencies	Inner RMSE	Outcome
Without pairwise initialization	0	5.0883	Base fallback
Pairwise initialization enabled	3	5.0811	Base+FDHG

Table 11: **Greedy screening is stable across Base feature budgets.** Across 16 task–budget settings, Greedy screening retains at least one dependency in 15; by the screening rule, each retained set improves the train-only inner-fold score over the corresponding Base. The remaining setting falls back to the base representation.

Task	**Budgets tested**	**FDHG selected**	**Selected improvements**
user-attendance	4	3 / 4	3 / 3
driver-position	4	4 / 4	4 / 4
studies-enrollment	4	4 / 4	4 / 4
study-outcome	4	4 / 4	4 / 4
Total	**16**	**15 / 16**	**15 / 15**

With pairwise rescue enabled, the pair `standings:wins`→`points` and `results:positionOrder`→`points` passes the train-only pairwise criterion and initializes Greedy selection. Subsequent Greedy expansion produces a final set of three selected dependencies and reduces the inner RMSE to 5.0811.

The pairwise-rescue ablation isolates a case in which singleton screening alone is insufficient. The result does not imply that pairwise initialization is universally necessary; rather, it provides a conservative recovery mechanism for complementary dependency signals that become useful only when considered jointly. Because the rescue decision uses exclusively the same train-only temporal evidence as singleton screening, it preserves the leakage-safe selection protocol used throughout FDHG.

C.1 Sensitivity to the Base Feature Budget

We evaluate whether FDHG remains useful as the underlying base representation changes in size. For four representative tasks, we vary the base feature budget over $\{4, 8, 12, 16\}$ and rerun train-only FDHG selection under the same protocol.

FDHG is selected in 15 of the 16 task-budget settings and improves over the corresponding base representation in every selected case. At the smallest budget for `user-attendance`, no FDHG augmentation passes the train-only selection criterion, so the procedure conservatively falls back to the base representation. These inner-fold results characterize the selection procedure rather than out-of-sample gains.

C.2 Independent vs. Greedy Selection

Independent and Greedy differ in how they assess dependency utility: Independent evaluates each dependency in isolation relative to the base representation, whereas Greedy evaluates its conditional gain given the dependencies already selected.

We compare Independent and Greedy on the 15-task common support defined by tasks for which both selectors produce non-empty FDHG augmentations. All comparisons use the same frozen candidate pool, base representation, temporal folds, HGB selection decoder, and predefined train-only selection tolerance.

Table 12: **Greedy is never worse than Independent beyond the predefined train-only tolerance and usually produces a sparser FDHG representation.** Results are reported on the 15-task common support, defined as tasks for which both selectors produce non-empty FDHG augmentations. Outcome categories are mutually exclusive.

Train-only outcome	**# Tasks**
Greedy better beyond tolerance	**6**
Within tolerance (non-exact)	7
Exact tie	2
Independent better beyond tolerance	**0**
Greedy uses fewer dependencies	**11 / 15**
Same number of dependencies	2 / 15
Independent uses fewer dependencies	2 / 15
Median reduction with Greedy	**7 dependencies**

Greedy improves beyond the selection tolerance on 6 of the 15 tasks and is never worse than Independent beyond that tolerance. On the remaining tasks, the two selectors are either within tolerance or exactly tied. Greedy also selects fewer dependencies on 11 of 15 tasks, with a median reduction of seven dependencies.

Overall, conditional selection provides a favorable accuracy–complexity trade-off by recovering additional utility when dependency interactions matter while avoiding redundant FDHG blocks when the two selectors perform similarly.

C.3 Random-K Dependency-Selection Ablation

To test whether FDHG gains arise from selecting informative dependencies rather than simply adding more dependency blocks, we compare Greedy selection with the base representation without FDHG augmentation, using all candidate dependencies, and Random-K, which samples the same number of dependencies selected by Greedy. For Random-K, we evaluate 20 independently sampled subsets per task.

Table 13: **On train-only inner folds, Greedy selection scores above cardinality-matched random selection.** Random-K reports mean $\pm$ standard deviation over 20 random subsets containing the same number of dependencies as Greedy. None of the 80 random subsets scores above the corresponding Greedy-selected set. All scores are computed on the same inner folds used by Greedy selection.

Task	Metric	Base	All	Random-K	Greedy (ours)
user-attendance	RMSE ↓	0.7003	0.6985	0.6989 ± 0.0012	**0.6960**
driver-position	RMSE ↓	5.0883	5.1863	5.1602 ± 0.0215	**5.0811**
studies-enrollment	RMSE ↓	107200.45	107406.27	107215.37 ± 25.62	**107179.84**
study-outcome	AUROC ↑	0.5820	0.5928	0.5861 ± 0.0044	**0.5979**

Greedy achieves the best train-only score on all four representative tasks. None of the 80 cardinality-matched Random-K subsets outperforms its corresponding Greedy-selected representation. Using all candidate dependencies is also unreliable, degrading performance relative to the base representation on `driver-position` and `studies-enrollment`. Because Greedy is selected on these same inner folds, this comparison shows that the selector finds better subsets than random choice under its own objective; out-of-sample effects are reported in Tables 3 and 4.

C.4 Efficiency

FDHG introduces additional computation because candidate dependency blocks must be constructed and, for the train-only selectors, repeatedly evaluated across inner temporal folds. We therefore compare Canonical DFS, Base, Base+All FDHG, Independent, and Greedy on four representative tasks under the same experimental configuration.

Table 14: **Efficiency of the relational feature-construction strategies.** Values are mean ± standard deviation across four representative tasks. Feature count denotes the number of downstream model-input columns, whereas FDHG dependency count denotes logical dependency blocks. Wall-clock time covers the complete strategy run in this audit, including feature construction, train-only selection when applicable, and the forced final evaluation. Peak memory is maximum resident set size (RSS).

Strategy	# Features	# FDHG deps.	Time (s)	Peak RSS (GiB)
Canonical DFS	6.50 ± 3.79	–	29.34 ± 24.22	4.16 ± 3.41
Base	6.25 ± 2.63	–	33.90 ± 31.23	4.12 ± 3.40
Base+All FDHG	278.00 ± 55.03	32.00 ± 0.00	224.06 ± 281.19	4.51 ± 3.21
Independent	17.50 ± 21.11	1.75 ± 3.50	160.79 ± 73.28	4.87 ± 4.22
Greedy	30.75 ± 11.44	3.00 ± 1.41	4720.14 ± 6105.00	4.39 ± 3.11

Median wall-clock times are 24.41 s, 25.49 s, 111.65 s, 148.60 s, and 2316.29 s for Canonical DFS, Base, Base+All FDHG, Independent, and Greedy, respectively.

Canonical DFS and Base have similar runtime and feature dimensionality in this four-task audit. Independent incurs additional computation from evaluating candidate dependency augmentations, while remaining substantially cheaper than Greedy. Greedy is considerably more expensive because its conditional search repeatedly evaluates remaining candidates across the inner temporal folds.

The additional search cost does not translate into a comparably large persistent representation. Greedy retains only 3.0 FDHG dependencies on average, compared with all 32 candidate dependencies in Base+All FDHG. Because each dependency can emit multiple statistical and missingness columns, the number of logical dependencies and the resulting model-input feature dimensionality are not identical.

Peak RSS is broadly comparable across strategies in this four-task audit, whereas wall-clock time varies substantially by task. Thus, the principal computational cost of Greedy is repeated conditional evaluation during offline representation search rather than the dimensionality of the frozen representation.

D Additional Experimental Results

D.1 Full Relational Compiler Comparison

Table 15 reports the complete task-level comparison underlying the aggregate relational feature-compiler results in Section 5.2. We report mean ± sample standard deviation over seeds 41–44 only when all four evaluation seeds completed successfully.

Table 15: **Full task-level comparison of relational feature compilers.** Results are reported across the 51-task benchmark under the common TabPFN evaluation protocol used in Section 5.2. Values are mean ± sample standard deviation over seeds 41–44 and are shown only when all four evaluation seeds completed successfully. Matched DFS, JUICE, and FastProp use the budget-controlled representations used in the main comparison, whereas FastDFS uses its materialized representation. A dash denotes an unavailable complete four-seed evaluation. Complete four-seed coverage is 41 tasks for Matched DFS, 38 for JUICE, 36 for FastProp, 27 for FastDFS, and 50 for DARC.

Dataset	Task	Metric	Matched DFS	JUICE	FastProp	FastDFS	DARC
dbinfer-avs	repeater	AUROC ↑	–	–	–	–	0.5428 ± 0.0161
dbinfer-retailrocket	cvr	AUROC ↑	–	–	–	–	0.8020 ± 0.0152
rel-amazon	item-churn	AUROC ↑	0.7519 ± 0.0194	0.7308 ± 0.0162	0.7795 ± 0.0162	0.7441 ± 0.0187	0.7880 ± 0.0160
rel-amazon	item-ltv	RMSE ↓	401.7228 ± 296.9142	395.6661 ± 294.9840	406.8766 ± 295.6378	407.2505 ± 287.8509	399.4721 ± 297.2065
rel-amazon	review-rating	RMSE ↓	0.9594 ± 0.0130	–	–	–	0.9616 ± 0.0114
rel-amazon	user-churn	AUROC ↑	0.6444 ± 0.0245	0.6084 ± 0.0223	0.6373 ± 0.0179	0.6120 ± 0.0229	0.6558 ± 0.0194
rel-amazon	user-ltv	RMSE ↓	65.3610 ± 67.4093	65.6505 ± 66.5683	65.2060 ± 65.4755	65.1368 ± 65.5198	64.4909 ± 67.6092
rel-arxiv	author-publication	RMSE ↓	1.0605 ± 0.0589	1.0132 ± 0.0804	1.0791 ± 0.0701	1.0590 ± 0.0575	0.9609 ± 0.0628
rel-arxiv	paper-citation	AUROC ↑	0.7711 ± 0.0124	0.7347 ± 0.0064	0.7494 ± 0.0115	0.8135 ± 0.0118	0.8062 ± 0.0121
rel-avito	ad-ctr	RMSE ↓	–	0.0996 ± 0.0001	–	0.0904 ± 0.0001	0.0939 ± 0.0000
rel-avito	searchinfo-isuserloggedon	AUROC ↑	0.5364 ± 0.0205	0.4852 ± 0.0156	0.5362 ± 0.0116	–	0.5315 ± 0.0187
rel-avito	searchstream-click	AUROC ↑	0.4888 ± 0.0787	–	–	–	0.4027 ± 0.0705
rel-avito	user-clicks	AUROC ↑	0.6468 ± 0.0234	0.5130 ± 0.0110	0.6465 ± 0.0244	0.6387 ± 0.0211	0.6366 ± 0.0218
rel-avito	user-visits	AUROC ↑	0.6938 ± 0.0139	0.4862 ± 0.0272	0.6960 ± 0.0132	0.6969 ± 0.0141	0.6938 ± 0.0139
rel-event	event_interest-interested	AUROC ↑	0.4962 ± 0.0171	–	–	–	0.7027 ± 0.0063
rel-event	event_interest-not_interested	AUROC ↑	0.5226 ± 0.3023	–	–	–	0.8118 ± 0.0385
rel-event	user-attendance	RMSE ↓	0.5227 ± 0.0008	0.4836 ± 0.0008	0.4815 ± 0.0011	0.4815 ± 0.0026	0.4870 ± 0.0014
rel-event	user-ignore	AUROC ↑	0.8941 ± 0.0087	0.8855 ± 0.0079	0.8832 ± 0.0077	0.8042 ± 0.0077	0.8462 ± 0.0103
rel-event	user-repeat	AUROC ↑	0.7122 ± 0.0008	0.6468 ± 0.0041	0.7219 ± 0.0007	0.6951 ± 0.0257	0.7130 ± 0.0005
rel-event	users-birthyear	RMSE ↓	–	8.4803 ± 0.0132	8.7038 ± 0.0146	–	8.7418 ± 0.0125
rel-f1	driver-dnf	AUROC ↑	0.6738 ± 0.0062	0.6526 ± 0.0095	0.7690 ± 0.0040	0.6431 ± 0.0197	0.7800 ± 0.0011
rel-f1	driver-position	RMSE ↓	4.6329 ± 0.0123	4.0087 ± 0.0076	3.8000 ± 0.0098	4.1294 ± 0.0150	3.6330 ± 0.0016
rel-f1	driver-top3	AUROC ↑	0.7541 ± 0.0051	0.7814 ± 0.0091	0.8727 ± 0.0025	0.8051 ± 0.0012	0.8731 ± 0.0006
rel-f1	qualifying-position	RMSE ↓	6.0558 ± 0.0000	6.0837 ± 0.0021	4.9061 ± 0.0163	–	5.3051 ± 0.0130
rel-f1	results-position	RMSE ↓	5.1123 ± 0.0003	4.9337 ± 0.0111	3.2673 ± 0.0008	–	4.6583 ± 0.0063
rel-hm	item-sales	RMSE ↓	0.5755 ± 0.1808	0.6375 ± 0.1621	0.5819 ± 0.1732	0.5694 ± 0.1774	0.5116 ± 0.1715
rel-hm	transactions-price	RMSE ↓	0.0245 ± 0.0019	–	–	–	0.0245 ± 0.0019
rel-hm	user-churn	AUROC ↑	0.6744 ± 0.0149	0.5287 ± 0.0214	0.6958 ± 0.0096	0.6758 ± 0.0061	0.6742 ± 0.0137
rel-ratebeer	beer-churn	AUROC ↑	0.7602 ± 0.0159	0.5608 ± 0.0173	0.7278 ± 0.0144	0.8856 ± 0.0115	0.7900 ± 0.0159
rel-ratebeer	beer_ratings-total_score	RMSE ↓	–	0.7120 ± 0.0139	0.7069 ± 0.0128	–	0.6998 ± 0.0162
rel-ratebeer	brewer-dormant	AUROC ↑	0.7644 ± 0.0109	0.7891 ± 0.0084	0.7978 ± 0.0109	0.8083 ± 0.0075	0.8148 ± 0.0120
rel-ratebeer	user-churn	AUROC ↑	0.8359 ± 0.0141	0.8706 ± 0.0089	0.7950 ± 0.0137	0.9500 ± 0.0057	0.8355 ± 0.0132
rel-ratebeer	user-count	RMSE ↓	40.7993 ± 7.5571	42.0940 ± 8.2481	43.5688 ± 6.7248	37.7667 ± 8.0201	38.2153 ± 7.3608
rel-salt	item-incoterms	Accuracy ↑	0.6586 ± 0.0163	–	–	–	0.6643 ± 0.0156
rel-salt	item-plant	Accuracy ↑	–	–	–	–	0.6111 ± 0.0089
rel-salt	item-shippoint	Accuracy ↑	–	–	–	–	0.4674 ± 0.0111
rel-salt	sales-group	Accuracy ↑	–	–	–	–	–
rel-salt	sales-incoterms	Accuracy ↑	0.6934 ± 0.0228	0.6038 ± 0.0106	0.9971 ± 0.0009	–	0.6934 ± 0.0228

Continued on next page

Dataset	Task	Metric	Matched DFS	JUICE	FastProp	FastDFS	DARC
rel-salt	sales-office	Accuracy ↑	0.9994 ± 0.0005	0.9994 ± 0.0005	0.9994 ± 0.0005	–	0.9994 ± 0.0005
rel-salt	sales-payterms	Accuracy ↑	–	0.2956 ± 0.0168	0.3244 ± 0.0137	–	0.2955 ± 0.0158
rel-salt	sales-shipcond	Accuracy ↑	0.3121 ± 0.0073	0.2813 ± 0.0080	–	–	0.3097 ± 0.0101
rel-stack	post-votes	RMSE ↓	0.4099 ± 0.2125	0.4626 ± 0.2502	0.4085 ± 0.2110	0.4021 ± 0.1794	0.4117 ± 0.2165
rel-stack	user-badge	AUROC ↑	0.8285 ± 0.0401	–	–	0.8176 ± 0.0320	0.8453 ± 0.0378
rel-stack	user-engagement	AUROC ↑	0.8763 ± 0.0173	–	–	0.8456 ± 0.0120	0.9027 ± 0.0133
rel-trial	eligibilities-adult	AUROC ↑	0.4962 ± 0.0134	0.4757 ± 0.0162	0.5464 ± 0.0107	–	0.5552 ± 0.0257
rel-trial	eligibilities-child	AUROC ↑	0.4883 ± 0.0123	0.4972 ± 0.0165	0.6020 ± 0.0087	–	0.5300 ± 0.0146
rel-trial	site-success	RMSE ↓	0.4869 ± 0.0040	0.4861 ± 0.0029	0.4847 ± 0.0049	0.4870 ± 0.0040	0.4869 ± 0.0040
rel-trial	studies-enrollment	RMSE ↓	–	$64564.1679 \pm 21313.0734$	$63798.7060 \pm 20832.8339$	–	$63780.2962 \pm 20829.9328$
rel-trial	studies-has_dmc	AUROC ↑	0.5625 ± 0.0119	0.4929 ± 0.0137	0.6762 ± 0.0152	–	0.6907 ± 0.0084
rel-trial	study-adverse	RMSE ↓	369.2789 ± 140.2798	362.0842 ± 142.3368	345.1714 ± 143.4760	320.7027 ± 154.2480	343.8705 ± 145.4089
rel-trial	study-outcome	AUROC ↑	0.5451 ± 0.0025	0.6547 ± 0.0026	0.5984 ± 0.0039	0.6604 ± 0.0006	0.6061 ± 0.0024

D.2 Feature Counts and Coverage

Budget matching for external feature compilers. For each task, let K_i denote the realized number of model-input columns in the frozen DARC representation. We use K_i as the target budget for the budget-controlled external feature compilers. For JUICE, metadata columns used only for alignment or identification are excluded, and generated feature columns are retained in deterministic export order up to K_i. FastProp is constructed under the same task-specific feature cap and may realize fewer than K_i usable model-input columns. Canonical DFS uses the atomic declaration-level matching procedure described in Appendix B.2.

Table 16 reports the realized dimensionality of the relational representations used in the compiler comparison. Feature counts refer to downstream model-input columns rather than logical feature declarations or dependency blocks.

Table 16: **Task-level feature counts for relational feature compilers.** Counts denote realized downstream model-input columns in the representation used for each compiler comparison. Matched DFS, JUICE, and FastProp use the budget-controlled representations described in Section 5.2, whereas FastDFS uses its materialized representation and DARC uses the representation frozen by the train-only gate. A dash denotes an unavailable representation artifact.

Dataset	Task	Matched DFS	JUICE	FastProp	FastDFS	DARC
dbinfer-avs	repeater	2	–	–	–	4
dbinfer-retailrocket	cvr	2	–	–	–	4
rel-amazon	item-churn	4	5	4	10	4
rel-amazon	item-ltv	4	5	4	10	4
rel-amazon	review-rating	2	–	–	–	1
rel-amazon	user-churn	4	5	4	9	4
rel-amazon	user-ltv	4	5	4	9	4
rel-arxiv	author-publication	2	3	2	1	2
rel-arxiv	paper-citation	4	7	6	9	6
rel-avito	ad-ctr	2	2	–	35	1
rel-avito	searchinfo-isuserloggedon	4	2	4	29	4
rel-avito	searchstream-click	6	–	–	–	8
rel-avito	user-clicks	4	6	5	23	5
rel-avito	user-visits	2	2	1	23	1
rel-event	event_interest-interested	12	–	–	–	13
rel-event	event_interest-not_interested	12	–	–	–	15
rel-event	user-attendance	12	30	30	437	30
rel-event	user-ignore	14	29	31	437	31
rel-event	user-repeat	14	15	14	437	14
rel-event	users-birthyear	2	2	1	437	1
rel-f1	driver-dnf	12	26	27	63	27
rel-f1	driver-position	4	34	35	63	35
rel-f1	driver-top3	12	34	35	63	35
rel-f1	qualifying-position	4	2	5	5	5
rel-f1	results-position	4	2	6	13	6
rel-hm	item-sales	6	8	7	21	7
rel-hm	transactions-price	2	–	–	–	2
rel-hm	user-churn	4	5	4	14	4
rel-ratebeer	beer-churn	8	7	8	95	8
rel-ratebeer	beer_ratings-total_score	6	2	16	85	17
rel-ratebeer	brewer-dormant	14	19	24	120	24
rel-ratebeer	user-churn	14	27	27	91	27
rel-ratebeer	user-count	6	7	6	91	7
rel-salt	item-incoterms	2	–	–	16	2
rel-salt	item-plant	2	–	–	16	2
rel-salt	item-shippoint	2	–	–	16	2
rel-salt	sales-group	2	–	–	17	1

Continued on next page

Dataset	Task	Matched DFS	JUICE	FastProp	FastDFS	DARC
rel-salt	sales-incoterms	14	15	14	17	14
rel-salt	sales-office	14	15	8	17	14
rel-salt	sales-payterms	2	3	1	17	2
rel-salt	sales-shipcond	2	2	–	17	1
rel-stack	post-votes	4	11	14	31	14
rel-stack	user-badge	6	–	–	30	37
rel-stack	user-engagement	6	–	–	30	7
rel-trial	eligibilities-adult	4	2	4	23	5
rel-trial	eligibilities-child	4	2	7	23	14
rel-trial	site-success	2	2	1	1	1
rel-trial	studies-enrollment	2	2	4	86	4
rel-trial	studies-has_dmc	4	2	62	86	67
rel-trial	study-adverse	4	5	2	86	4
rel-trial	study-outcome	4	43	56	86	56

D.3 Strict Common-Support Feature-Engineering Comparison

To verify that the aggregate conclusions are not driven by comparator-specific task coverage, we additionally evaluate all feature-engineering methods on the strict intersection of tasks for which our representation, JUICE, FastProp, and matched Canonical DFS are simultaneously available. This yields a common support of 32 tasks.

The strict common-support analysis preserves the same qualitative conclusion as the pairwise comparisons in Section 5.2: our selected representation achieves a positive win–loss margin against each relational feature-engineering baseline even when all methods are restricted to exactly the same set of tasks.

D.4 Train-Only Gate Oracle Diagnostic

As a diagnostic, we compare the representation selected from inner temporal training folds with a hindsight oracle over available official-validation representations. Excluding four tasks with only one available candidate, the gate selects the hindsight-best representation on 28/47 tasks (59.6%), lies within the predefined task-specific tolerance on 34/47 (72.3%), and has median direction-aligned oracle regret of zero.

Because non-selected Canonical DFS validation scores are unavailable for some tasks, this is an available-candidate diagnostic rather than a complete four-way oracle analysis. Official-validation performance is used only for this post hoc diagnostic and never influences feature or strategy selection.

Table 17: **Strict common-support relational feature-engineering comparison.** All methods are evaluated on the identical subset of 32 tasks jointly supported by our method, JUICE, FastProp, and matched Canonical DFS.

Comparator	Wins	Ties	Losses	W–L
JUICE	**25**	1	6	**+19**
FastProp	**18**	1	13	**+5**
Canonical DFS (matched)	**22**	4	6	**+16**

D.5 Representative Representation and Decoder Results

Table 18 reports representative task-level representation comparisons and downstream-decoder transfer results.

Table 18: **Representative predictive results and decoder generalization.** The left block compares candidate relational representations using TabPFN. The right block evaluates the representation selected *beforehand* by the train-only gate using heterogeneous downstream decoders. Results are mean ± sample standard deviation over seeds 41-44 unless otherwise specified. Bold denotes the best TabPFN representation in hindsight for each task, while † denotes the representation selected using train-only temporal evidence. The selected representation is frozen and is not reselected for individual downstream decoders.

			Representation comparison (TabPFN)					**Frozen selected representation**					
Dataset	Task	Metric	DFS	Base	+FDHG Ind.	+FDHG Greedy	Selected	TabPFN	XGBoost	CatBoost	HGB	TabICL	RealMLP
rel-f1	driver-top3	AUROC ↑	0.7541 ± 0.0051	0.8361 ± 0.0022	$0.8361 \pm 0.0022^{*}$	$\mathbf{0.8731 \pm 0.0006}^{\dagger}$	Greedy	0.8731 ± 0.0006	0.8255 ± 0.0033	0.8569 ± 0.0024	0.8548	0.8728 ± 0.0013	0.8572 ± 0.0134
rel-f1	driver-position	RMSE ↓	4.6329 ± 0.0123	3.6970 ± 0.0020	$3.6970 \pm 0.0020^{*}$	$\mathbf{3.6330 \pm 0.0016}^{\dagger}$	Greedy	3.6330 ± 0.0016	4.0457 ± 0.0263	3.6442 ± 0.0114	3.6967	3.6708 ± 0.0058	3.5904 ± 0.0214
rel-stack	user-badge	AUROC ↑	0.8285 ± 0.0401	0.8428 ± 0.0359	0.8452 ± 0.0391	$\mathbf{0.8453 \pm 0.0378}^{\dagger}$	Greedy	0.8453 ± 0.0378	0.8583 ± 0.0002	0.8605 ± 0.0002	0.8601	0.8046 ± 0.0315	0.8287 ± 0.0408
rel-event	user-attendance	RMSE ↓	0.5228 ± 0.0007	0.4903 ± 0.0020	$\mathbf{0.4819 \pm 0.0017}$	$0.4870 \pm 0.0014^{\dagger}$	Greedy	0.4870 ± 0.0014	0.5272 ± 0.0020	0.4900 ± 0.0007	0.4894	0.4996 ± 0.0051	0.4971 ± 0.0042
rel-trial	studies-has_dmc	AUROC ↑	0.5625 ± 0.0119	0.6670 ± 0.0147	$\mathbf{0.6920 \pm 0.0101}$	$0.6907 \pm 0.0084^{\dagger}$	Greedy	0.6907 ± 0.0084	0.6966 ± 0.0009	0.7035 ± 0.0006	0.7036	0.6753 ± 0.0190	0.6831 ± 0.0116
rel-ratebeer	brewer-dormant	AUROC ↑	0.7644 ± 0.0109	$\mathbf{0.8179 \pm 0.0121}$	$0.8148 \pm 0.0120^{\dagger}$	–	Independent	0.8148 ± 0.0120	0.8043 ± 0.0005	0.8075 ± 0.0001	0.8075	0.7852 ± 0.0104	0.8066 ± 0.0100
rel-stack	user-engagement	AUROC ↑	0.8763 ± 0.0173	$0.9027 \pm 0.0133^{\dagger}$	0.9013 ± 0.0123	$\mathbf{0.9034 \pm 0.0132}$	Base	0.9027 ± 0.0133	0.8969 ± 0.0001	0.8987 ± 0.0001	0.8992	0.8991 ± 0.0192	0.8913 ± 0.0180
rel-event	user-repeat	AUROC ↑	$\mathbf{0.7130 \pm 0.0005}^{\dagger}$	0.6967 ± 0.0007	0.6807 ± 0.0036	0.6944 ± 0.0013	DFS	0.7130 ± 0.0005	0.6631 ± 0.0002	0.6891 ± 0.0093	0.6867	0.7148 ± 0.0027	0.7005 ± 0.0117

TabPFN, XGBoost, CatBoost, TabICL, and RealMLP report mean ± sample standard deviation over seeds 41–44 where available. HGB reports one deterministic evaluation on the same frozen representation. *Independent yields no usable FD features at final refit on the marked tasks and therefore reduces to the dependency-agnostic base representation. A dash in the representation-comparison block indicates that the corresponding distinct representation is unavailable in the current canonical artifact. Full task-level decoder results are reported in Appendix E.

E COMPLETE TASK-LEVEL DECODER RESULTS

Table 19 reports the complete task-level decoder evaluation for all 51 benchmark tasks. For each task, we first freeze the representation selected by the train-only compiler gate and then evaluate that same representation with each downstream decoder; representation selection is never repeated using official validation.

Coverage is 50/51 tasks for TabPFN (200 runs), 50/51 for XGBoost (200 runs), 48/51 for CatBoost (192 runs), 51/51 for HGB (51 deterministic evaluations), 49/51 for TabICL (196 runs), and 51/51 for RealMLP (204 runs). Multi-seed results report mean $\pm$ sample standard deviation over seeds 41–44. HGB is deterministic and is therefore reported as a single evaluation. A dash denotes an unavailable evaluation in the corresponding decoder inventory; `rel-salt/sales-group` is explicitly unsupported by TabPFN.

Table 19: **Complete task-level decoder results (Part 1/2).** Each decoder is evaluated on the same representation selected by the train-only compiler gate.

Dataset	Task	Metric	Selected	TabPFN	XGBoost	CatBoost	HGB	TabICL	RealMLP
dbinfer-avs	repeater	AUROC	Base	0.542773 ± 0.016108	0.535278 ± 0.000989	0.543948 ± 0.000153	0.541517	0.534603 ± 0.013302	0.524239 ± 0.019583
dbinfer-retailrocket	cvr	AUROC	Base	0.802000 ± 0.015235	0.791160 ± 0.001285	0.823004 ± 0.001225	0.820754	0.799075 ± 0.009847	0.777895 ± 0.011550
rel-amazon	item-churn	AUROC	Base	0.788021 ± 0.016022	0.790633 ± 0.000012	0.791317 ± 0.000060	0.791105	0.788098 ± 0.016683	0.782523 ± 0.018513
rel-amazon	item-ltv	RMSE	Base	$399.472091 \pm 297.206546$	471.371796 ± 1.470782	450.562010 ± 1.134572	436.590289	$416.997275 \pm 285.955565$	$424.099786 \pm 280.002795$
rel-amazon	review-rating	RMSE	Base	0.961604 ± 0.011407	0.969560 ± 0.000026	0.969555 ± 0.000003	0.969553	0.967875 ± 0.011179	0.967486 ± 0.009426
rel-amazon	user-churn	AUROC	Base	0.655833 ± 0.019422	0.668383 ± 0.000059	0.668789 ± 0.000027	0.668559	0.652903 ± 0.017771	0.655981 ± 0.018191
rel-amazon	user-ltv	RMSE	Base	64.490929 ± 67.609245	42.044184 ± 0.057078	41.755919 ± 0.017955	41.685034	64.639578 ± 66.987316	65.619798 ± 66.364663
rel-arxiv	author-publication	RMSE	Base	0.960917 ± 0.062835	1.034035 ± 0.001068	0.947969 ± 0.004965	0.957025	1.047573 ± 0.134847	0.941140 ± 0.055425
rel-arxiv	paper-citation	AUROC	Base	0.806153 ± 0.012095	0.799695 ± 0.000078	0.803119 ± 0.000016	0.803295	0.796358 ± 0.016078	0.801542 ± 0.011038
rel-avito	ad-ctr	RMSE	Base	0.093946 ± 0.000027	0.093377 ± 0.000029	0.093303 ± 0.000011	0.093377	0.093061 ± 0.000000	0.093859 ± 0.000655
rel-avito	searchinfo-isuserloggedon	AUROC	Base	0.531482 ± 0.018666	0.530686 ± 0.000202	0.539005 ± 0.000167	0.539532	0.525303 ± 0.010116	0.519061 ± 0.025547
rel-avito	searchstream-click	AUROC	Base	0.402732 ± 0.070511	0.501083 ± 0.000431	0.500138 ± 0.001245	0.499431	0.445875 ± 0.054091	0.471726 ± 0.097032
rel-avito	user-clicks	AUROC	Base	0.636634 ± 0.021788	0.581969 ± 0.002046	0.604876 ± 0.003601	0.622480	0.625045 ± 0.028749	0.624510 ± 0.020313
rel-avito	user-visits	AUROC	Base	0.693811 ± 0.013874	0.687729 ± 0.000109	0.689307 ± 0.000074	0.688186	0.692027 ± 0.014121	0.690751 ± 0.012165
rel-event	event_interest-interested	AUROC	Greedy	0.702742 ± 0.006343	0.699352 ± 0.002136	0.706491 ± 0.000248	0.703727	0.686738 ± 0.007338	0.678751 ± 0.016344
rel-event	event_interest-not_interested	AUROC	Greedy	0.811770 ± 0.038516	0.819303 ± 0.007022	0.830697 ± 0.001522	0.832392	0.833333 ± 0.013824	0.771751 ± 0.026755
rel-event	user-attendance	RMSE	Greedy	0.486959 ± 0.001420	0.527155 ± 0.001961	0.489962 ± 0.000711	0.489359	0.499646 ± 0.005060	0.497085 ± 0.004215
rel-event	user-ignore	AUROC	Greedy	0.846187 ± 0.010316	0.852622 ± 0.000112	0.852258 ± 0.000092	0.851613	0.846315 ± 0.010169	0.846151 ± 0.010094
rel-event	user-repeat	AUROC	DFS	0.712974 ± 0.000460	0.663085 ± 0.000184	0.689061 ± 0.009330	0.686706	0.714827 ± 0.002744	0.700516 ± 0.011716
rel-event	users-birthyear	RMSE	Base	8.741848 ± 0.012498	8.673759 ± 0.000437	8.673430 ± 0.000227	8.673020	8.699741 ± 0.023800	8.646834 ± 0.047013
rel-f1	driver-dnf	AUROC	Greedy	0.779950 ± 0.001061	0.744626 ± 0.005483	0.757333 ± 0.004255	0.768218	0.776213 ± 0.001423	0.772422 ± 0.005923
rel-f1	driver-position	RMSE	Greedy	3.632961 ± 0.001610	4.045675 ± 0.026346	3.644156 ± 0.011420	3.696733	3.670796 ± 0.005844	3.590404 ± 0.021400
rel-f1	driver-top3	AUROC	Greedy	0.873121 ± 0.000628	0.825527 ± 0.003293	0.856910 ± 0.002390	0.854840	0.872796 ± 0.001259	0.857215 ± 0.013426
rel-f1	qualifying-position	RMSE	Base	5.305092 ± 0.012952	5.764184 ± 0.021913	5.339084 ± 0.019051	5.352265	5.424415 ± 0.027104	5.245484 ± 0.279616
rel-f1	results-position	RMSE	Base	4.658318 ± 0.006347	4.518223 ± 0.003876	4.365414 ± 0.021170	4.402860	4.570665 ± 0.015664	4.445394 ± 0.074115
rel-hm	item-sales	RMSE	Base	0.511571 ± 0.171529	0.454678 ± 0.000479	0.453715 ± 0.000556	0.461126	0.460431 ± 0.170478	0.524524 ± 0.178847

Table 20: **Complete task-level decoder results (continued).**

Dataset	Task	Metric	Selected	TabPFN	XGBoost	CatBoost	HGB	TabICL	RealMLP
rel-hm	transactions-price	RMSE	Base	0.024503 ± 0.001921	0.023476 ± 0.000000	0.023454 ± 0.000002	0.023477	0.024254 ± 0.001925	0.024316 ± 0.001969
rel-hm	user-churn	AUROC	Base	0.674241 ± 0.013673	0.688961 ± 0.000079	0.690066 ± 0.000049	0.689223	0.673716 ± 0.013201	0.671272 ± 0.012510
rel-ratebeer	beer-churn	AUROC	Base	0.790043 ± 0.015937	0.793220 ± 0.000048	0.793213 ± 0.000043	0.792892	0.789122 ± 0.015743	0.785908 ± 0.019272
rel-ratebeer	beer_ratings-total_score	RMSE	Greedy	0.699752 ± 0.016152	–	–	0.698830	0.700796 ± 0.016341	0.707804 ± 0.015411
rel-ratebeer	brewer-dormant	AUROC	Independent	0.814799 ± 0.011954	0.804258 ± 0.000535	0.807514 ± 0.000057	0.807488	0.785179 ± 0.010403	0.806555 ± 0.009995
rel-ratebeer	user-churn	AUROC	Independent	0.835507 ± 0.013198	0.830323 ± 0.000125	0.833045 ± 0.000128	0.834302	0.836866 ± 0.012543	0.834183 ± 0.012545
rel-ratebeer	user-count	RMSE	Base	38.215337 ± 7.360831	36.280015 ± 0.098843	34.525706 ± 0.326689	36.023310	36.362854 ± 7.173828	38.548049 ± 7.066473
rel-salt	item-incoterms	Accuracy	Base	0.664250 ± 0.015602	0.668893 ± 0.000217	0.673159 ± 0.003057	0.660486	0.665875 ± 0.015612	0.662250 ± 0.012990
rel-salt	item-plant	Accuracy	Base	0.611125 ± 0.008929	0.609469 ± 0.000000	–	0.609469	–	0.611125 ± 0.008929
rel-salt	item-shippoint	Accuracy	Base	0.467375 ± 0.011063	0.467346 ± 0.000000	–	0.467346	–	0.139875 ± 0.001652
rel-salt	sales-group	Accuracy	Base	*Unsupported*	0.067797 ± 0.000000	0.067797 ± 0.000000	0.067797	0.067875 ± 0.008440	0.045875 ± 0.031705
rel-salt	sales-incoterms	Accuracy	DFS	0.693375 ± 0.022761	0.675909 ± 0.000485	0.670956 ± 0.000712	0.647181	0.721250 ± 0.013181	0.627875 ± 0.014020
rel-salt	sales-office	Accuracy	DFS	0.999375 ± 0.000479	0.999003 ± 0.000013	0.999063 ± 0.000000	0.998349	0.999375 ± 0.000479	0.999375 ± 0.000479
rel-salt	sales-payterms	Accuracy	Base	0.295500 ± 0.015801	0.293793 ± 0.000011	0.293923 ± 0.000046	0.206305	0.294875 ± 0.016484	0.294625 ± 0.016815
rel-salt	sales-shipcond	Accuracy	Base	0.309750 ± 0.010087	0.307116 ± 0.000014	0.307067 ± 0.000091	0.307123	0.309250 ± 0.008170	0.309500 ± 0.008916
rel-stack	post-votes	RMSE	Base	0.411737 ± 0.216541	0.287954 ± 0.000480	0.285100 ± 0.000589	0.286883	0.408914 ± 0.205076	0.435081 ± 0.230382
rel-stack	user-badge	AUROC	Greedy	0.845258 ± 0.037811	0.858334 ± 0.000151	0.860509 ± 0.000159	0.860149	0.804614 ± 0.031522	0.828729 ± 0.040830
rel-stack	user-engagement	AUROC	Base	0.902698 ± 0.013322	0.896902 ± 0.000141	0.898723 ± 0.000113	0.899228	0.899068 ± 0.019249	0.891253 ± 0.017959
rel-trial	eligibilities-adult	AUROC	Base	0.555244 ± 0.025727	0.541665 ± 0.000651	0.549270 ± 0.001012	0.549808	0.539918 ± 0.016608	0.545608 ± 0.020976
rel-trial	eligibilities-child	AUROC	Greedy	0.530021 ± 0.014596	0.540495 ± 0.000480	0.544067 ± 0.000986	0.544064	0.508712 ± 0.030521	0.530322 ± 0.006401
rel-trial	site-success	RMSE	Base	0.486923 ± 0.004012	0.482624 ± 0.000009	0.482570 ± 0.000006	0.482618	0.484304 ± 0.003838	0.482574 ± 0.003764
rel-trial	studies-enrollment	RMSE	Base	$63780.296183 \pm 20829.932803$	$125834.580799 \pm 1.129434$	$125834.379865 \pm 0.000036$	125847.268505	$63689.895904 \pm 20832.385274$	$63806.278773 \pm 20959.946199$
rel-trial	studies-has_dmc	AUROC	Greedy	0.690748 ± 0.008428	0.696554 ± 0.000878	0.703530 ± 0.000608	0.703620	0.675315 ± 0.018971	0.683105 ± 0.011587
rel-trial	study-adverse	RMSE	Base	$343.870467 \pm 145.408907$	362.375670 ± 0.026105	360.994043 ± 0.190102	364.545290	$330.472900 \pm 152.646688$	$350.494532 \pm 143.604732$
rel-trial	study-outcome	AUROC	Greedy	0.606062 ± 0.002357	0.579539 ± 0.004215	0.589135 ± 0.002658	0.592917	0.546519 ± 0.027942	0.598661 ± 0.011586

Coverage. TabPFN: 50/51 tasks (200 runs); XGBoost: 50/51 (200); CatBoost: 48/51 (192); HGB: 51/51 (51 deterministic evaluations); TabICL: 49/51 (196); RealMLP: 51/51 (204). TabPFN does not support `rel-salt/sales-group`. Unavailable evaluations are XGBoost on `rel-ratebeer/beer_ratings-total_score`; CatBoost on `rel-ratebeer/beer_ratings-total_score`, `rel-salt/item-plant`, and `rel-salt/item-shippoint`; and TabICL on `rel-salt/item-plant` and `rel-salt/item-shippoint`.