DO RDB FOUNDATION MODELS EVEN NEED DATA?

Linjie Xu, Yanlin Zhang, Quan Gan, Minjie Wang, & David Wipf *
University of Hong Kong, Shanghai X-Lab

ABSTRACT

Relational databases (RDBs) contain vast amounts of heterogeneous tabular information that can be exploited for predictive modeling purposes. Of course the space of potential targets is vast across enterprise settings, so it is preferable to avoid learning a new model each time there is a new estimation task. Foundation models based on in-context learning (ICL) offer a convenient option, but so far are mostly restricted to single-table operability, a presumed impediment being the difficulty in collecting or generating adequate RDB pre-training data. But is there any practical way around this bottleneck? We answer in the affirmative by demonstrating how already-existing single-table foundation models can be repurposed for RDBs when combined with a suitable class of relational encoder, such that no further pre-training or data collection is even required. This is possible because theoretical and empirical evidence suggests that ICL-specific encoder compression of variably-sized RDB neighborhoods should be constrained *within* high-dimensional RDB columns where all entities share units and roles, not *across* columns where the relevance of heterogeneous data types cannot be determined without label information. And conditioned on this particular restriction, encoder expressiveness is not compromised by excluding learnable parameters that would otherwise necessitate RDB data collection for pre-training. Practically, we develop scalable SQL primitives to implement the encoder stage within an open-source toolbox that achieves SOTA performance on new RDBs out of the box.

1 INTRODUCTION

Foundation models for tabular data, capable of handling new predictive tasks *without retraining*, are increasingly prevalent (Hollmann et al., 2025; Jingang et al., 2025; Zhang et al., 2025c;b). When predicated upon some form of in-context learning (ICL), these models push labeled instances from a previously unseen dataset through a single forward pass of a pre-trained Transformer architecture, and then output predictions at one or more user-specified testing points. Despite their promising performance thus far, these existing *single-table* foundation models do not address wide-ranging enterprise relational databases (RDBs) involving multiple inter-connected tables (Garcia-Molina et al., 2009). For example, on e-commerce platforms candidate RDB prediction targets may cover future product purchases (Ni et al., 2019), customer retention (Dave et al., 2014), click-through rates (mjkistler et al., 2016; Zykov et al., 2022), user churn (Ni et al., 2019), or charge/pre-payment attributes (Motl & Schulte, 2015). As reliance upon such capabilities continues to grow, moving beyond single-table solutions has become an important yet under-served frontier; see Appendix A.

A key bottleneck to such multi-table advancements is the presumed dependency on large volumes of mostly-unavailable RDB pre-training data, particularly for common numerically-heavy use cases that cannot be directly addressed with LLM prompting. This data paucity stems from both the difficulty in generating realistic synthetic RDBs, and the tendency of real-world enterprise RDBs to remain siloed by privacy/IP concerns. But is collecting multi-table relational data strictly necessary in the first place? Perhaps counter-intuitively, we find a viable path towards RDB foundation models *without* any data collection by re-examining the underpinnings of RDB predictive modeling.

Here intuition suggests that a rich parameterized encoder is paramount to accommodate RDB use cases, converting variably-sized RDB neighborhoods (possibly expressed as subgraphs), into dense fixed-length embeddings that mirror single-table settings. Indeed, most existing RDB predictive models trained via *per-dataset supervision* operate more-or-less in this fashion (Dwivedi et al., 2025; Robinson et al., 2024a; Wang et al., 2024), with a graph neural network (GNN) or Transformer-related architecture serving as the de facto encoder. These approaches have been shown to outperform alternatives based on combining parameter-free multi-table feature aggregation with trainable single-table prediction heads (Wang et al., 2024; Zhang et al., 2023b).

*Correspondence to davidwipf@gmail.com.

However, as we will argue both theoretically and empirically, what is natural in the supervised learning regime *need not necessarily transition to success in distinct ICL-based RDB foundation models*. The high-level rationale is that a dense encoder representation can interfere with the original column space of RDB tables, conflating units, roles, and levels of useful information. When a supervision signal is present this need not be problematic, since the encoder can simply learn to first prune away useless dimensions specific to a given dataset. But prior to seeing ICL samples within an RDB foundation model, a necessarily *fixed* encoder is incapable of adjudicating column roles, which may vary from dataset to dataset (or even task to task within a given RDB; see Figure 1). Hence we advocate for RDB encoders that only compress vertically *within* high-dimensional columns (where shared units facilitate interpretable aggregation), not horizontally *across* columns, where relevance is largely indeterminate without label information. Importantly, conditioned on this restriction to vertical compression, we formalize how a *parameter-free* encoder results in a minimal loss of expressiveness. And so we can directly pair this class of encoder with the most powerful existing single-table foundation models *with no training (or pre-training) required*. Circling back to our original question, the implication is that we do *not* actually need to collect or generate any RDB data, alleviating a non-trivial impediment to RDB foundation model development. In tracing this path, our main contributions distill as follows:

- We define a restricted class of RDB encoder that explicitly preserves column identities and interpretable information flow to a single-table ICL-prediction head. Within this class, we establish that encoder expressiveness is not compromised by excluding trainable parameters. This facilitates direct compatibility with the existing single-table foundation models while avoiding any reliance on problematic RDB pre-training data.
- We quantify how RDB encoders *outside* of the proposed class can provably increase estimation error and/or sample complexity when uninformative feature columns are present.
- Using scalable SQL primitives to implement the encoder stage, we introduce *RDBLearn*, an easy-to-use open-source RDB foundation model capable of handling completely new datasets out of the box with no training or fine-tuning whatsoever. The performance exceeds existing alternatives, including a non-reproducible, closed-source industry model that has been pre-trained with access to unknown real-world datasets. We also include ablations to support the conceptual underpinnings of the proposed pipeline.

We remark that parameter-free RDB encoding methods have been proposed in the past for engineering supervised predictive pipelines (Kanter & Veeramachaneni, 2015; Kramer et al., 2001; Zahradník et al., 2023). What differentiates our contribution is that we are not defaulting to such methods merely as a simplifying heuristic as in prior work. Instead, we are rigorously examining *why* a suitable family of parameter-free encoders may actually be *preferred* when the goal is ICL over unseen RDBs, particularly those with task-dependent partitions of useful and useless columns.

2 RDB Predictive Modeling

In a canonical supervised learning setting, we are given training data $\mathcal{D} = \{\boldsymbol{X}, \boldsymbol{y}\} \equiv \{\boldsymbol{x}_{i:}, y_i\}_{i=1}^{n}$ with instance feature rows $\boldsymbol{x}_{i:} \in \mathcal{X}^d$ and corresponding instance labels $y_i \in \mathcal{Y}$ for all i. The goal is then to learn a parameterized model f_θ such that $f_\theta(\boldsymbol{x}_{\text{test}}) \approx y_{\text{test}}$ at any test point $\{\boldsymbol{x}_{\text{test}}, y_{\text{test}}\}$, where in practice y_{test} is unknown. *RDB predictive modeling* generalizes the above via the inclusion of an additional set of auxiliary data tables $\mathcal{T} = \{\boldsymbol{T}^k\}_{k=1}^{K}$, where $\boldsymbol{T}^k \in \mathfrak{T}^{n_k \times d_k}$ denotes the k-th table associated with a given entity type. Each table row corresponds with a single instance of that entity (e.g., an individual user), and the columns encode instance attributes (e.g., elements of a user profile). These attributes are generally heterogeneous in nature, often consisting of continuous or discrete numerical values, categorical fields, text fragments, or temporal information. Note that without loss of generality, and for notational convenience later, we also assert that $\boldsymbol{T}^K \equiv \boldsymbol{X}$.

To complete its specification and make use of these auxiliary tables for making predictions, an RDB also includes a set of relations $\mathcal{R} = \{\boldsymbol{F}^k, \boldsymbol{p}^k\}_{k=1}^{K}$. Here each $\boldsymbol{p}^k \in \mathcal{P}^{n_k \times 1}$ denotes a primary key (PK) column, with elements uniquely identifying the rows of $\boldsymbol{T}^k$. Meanwhile, each column $\boldsymbol{f}_{:j}^k$ of $\boldsymbol{F}^k$ represents a foreign key (FK), whose elements are all given by values in some fixed PK column it references. In this way, the domain of every FK $\boldsymbol{f}_{:j}^k$ is some $\boldsymbol{p}^{k'}$.

2.1 A GENERIC RDB SUPERVISED LEARNING PIPELINE

A generic RDB predictive modeling pipeline predicated on *supervised learning* can be executed via the following steps:

1. Convert RDB $\{\mathcal{T}, \mathcal{R}\}$ as defined above to a heterogeneous graph $\mathcal{G}$. There are multiple ways of doing so (Wang et al., 2024), but the most common involves treating each table $\boldsymbol{T}^k$ as a node type and each row i within a given $\boldsymbol{T}^k$ as a node with features $\boldsymbol{t}^k_{i:}$. We then form directed edges using each FK column $\boldsymbol{f}^k_{:j}$ and PK column $\boldsymbol{p}^{k'}$ it points to within $\mathcal{R}$. This approach is widely adopted (Cvitkovic, 2020; Dwivedi et al., 2025; Fey et al., 2023; Zhang et al., 2023a;b).
2. Based on $\mathcal{G}$, sample H-hop subgraphs or ego-networks $\mathcal{G}_H(\boldsymbol{x}_{i:})$ that are centered at each target row $\boldsymbol{x}_{i:}$ within $\boldsymbol{X}$. For temporal RDBs, sampling should exclude nodes with timestamps later than $\boldsymbol{x}_{i:}$.
3. Independent of the original RDB or individual subgraph sizes, compute *fixed-length* embeddings $\boldsymbol{z}_{i:} = g_\phi[\mathcal{G}_H(\boldsymbol{x}_{i:})] \in \mathbb{R}^{d_z}$, where the *encoder* g_ϕ is some form of GNN or Transformer architecture.
4. Stack embeddings to form the revised $\mathcal{D} = \{\boldsymbol{Z}, \boldsymbol{y}\} \equiv \{\boldsymbol{z}_{i:}, y_i\}_{i=1}^n$ and train end-to-end by minimizing the supervised loss
$$\mathcal{L}^{\text{SL}}(\theta, \phi) = \sum_{i=1}^{n} -\log q_\theta\Big(y_i | g_\phi\big[\mathcal{G}_H(\boldsymbol{x}_{i:})\big]\Big) \tag{1}$$
over parameters ϕ from the encoder and θ from a suitable prediction head/decoder q_θ.
5. At inference time, update $\mathcal{G}$ to reflect any new collected data, including new unlabeled test rows of $\boldsymbol{X} = \boldsymbol{T}^K$, form the new subgraph $\mathcal{G}_H(\boldsymbol{x}_{\text{test}})$, and then compute $q_\theta\big(y_{\text{test}} | g_\phi\big[\mathcal{G}_H(\boldsymbol{x}_{\text{test}})\big]\big)$ for making predictions of y_{test}.

2.2 RDB FOUNDATION MODELS

Moving beyond the supervised learning setting of the previous section, the goal of ***RDB foundation models*** is to retain applicability across multiple RDBs, with minimal or no retraining required for each new predictive task. In addition to direct LLM-prompting approaches (Wydmuch et al., 2024), we discuss two notable possibilities that involve an explicit pre-training step over multiple RDBs, and therefore an inherent dependency on data collection.

Schema-agnostic models. Provided the encoder g_ϕ is designed to digest a broad spectrum of input RDB schema (across entity types and associated features) using a shared representational form, then it is possible to simultaneously train q_θ and/or g_ϕ over multiple real-world RDBs (Ranjan et al., 2025; Wang et al., 2025; Wu et al., 2025). The resulting model can, at least in principle, be applied to new unseen RDBs without retraining, or perhaps more realistically, with modest fine-tuning for any given task.

ICL-based models. Building on the growing development of models exploiting ICL for single-table data (Hollmann et al., 2025; Jingang et al., 2025; Zhang et al., 2025b), it is natural to consider extensions to multi-table RDBs using a graph encoder g_θ as adopted in previously-introduced supervised learning models (Fey et al., 2025). Expanding on step 4 from Section 2.1, the ICL multi-table training objective becomes

$$\mathcal{L}^{\text{ICL}}(\theta, \phi) = \mathbb{E}_{\mathcal{T},\mathcal{R}\sim p(\mathcal{T},\mathcal{R})}\Big[-\log q_\theta\Big(y_{\text{test}} | \boldsymbol{z}_{\text{test}}, \mathcal{D}\Big)\Big] \;\text{ with }\; \mathcal{D} = \{\boldsymbol{z}_{i:}, y_i\}_{i=1}^n,\;\; \boldsymbol{z} = g_\phi[\mathcal{G}_H(\boldsymbol{x})]. \tag{2}$$

Unlike in (1), the revised decoder module q_θ now consumes a set of labeled ICL samples $\{\boldsymbol{z}_{i:}, y_i\}_{i=1}^n$ as well as a test point $\boldsymbol{z}_{\text{test}} = g_\phi[\mathcal{G}_H(\boldsymbol{x}_{\text{test}})]$, and is charged with predicting y_{test}; see prior work for background justification of forming q_θ in this way for single tables (Hollmann et al., 2022; Nagler, 2023). All the requisite quantities are extracted from RDBs sampled during pre-training, which may be synthetically generated from some distribution $p(\mathcal{T}, \mathcal{R})$ and/or combined with available real-world RDBs. Synthetic generation has the advantage of unlimited volume, potentially as an extension of single-table synthetic-generation training pipelines already in common use (Hollmann et al., 2025; Jingang et al., 2025; Zhang et al., 2025b). In contrast, real-world RDB data with wide coverage is relatively difficult to collect, as most sources are private within enterprises.

At inference time, a *completely new* RDB $\{\mathcal{T}, \mathcal{R}\}$ is provided, including one or more unlabeled test query points $\boldsymbol{x}_{\text{test}}$ within $\boldsymbol{X} = \boldsymbol{T}^K$. Critically, *no further per-dataset training occurs*; instead, $\mathcal{D}$ and $\boldsymbol{z}_{\text{test}}$ are computed as in (2) using encoder g_ϕ, and then we form our final estimator as $q_\theta\big(y_{\text{test}}|\boldsymbol{z}_{\text{test}}, \mathcal{D}\big)$. In practice, the latter amounts to pushing $\boldsymbol{z}_{\text{test}}$ and the ICL samples within $\mathcal{D}$ through a single forward pass of a Transformer architecture used for instantiating q_θ.

2.3 Limitations

Thus far schema-agnostic models (Ranjan et al., 2025; Wang et al., 2025; Wu et al., 2025) have only been narrowly applied within small sets of RDBs for which varying degrees of pre-training and/or fine-tuning were applied. Hence their applicability outside of this regime on fundamentally different RDB types remains uncertain. Meanwhile for pure ICL-based models predicated on synthetic generation during pre-training, no existing open-source frameworks are actually available for transparent evaluation. We only have the unpublished *closed-source* KumoRFM approach (Fey et al., 2025) with key details of pre-training methodology and data generation missing; see Appendix A.

3 A Foundation for RDB Foundation Models

Given the established track record of ICL-based *single-table* foundation models, we intend to push similar principles into the more complex *multi-table* regime. In this regard, we will adopt an ICL-based prediction head q_θ, where ICL samples within a given dataset share a fixed dimension as in prior work. But we depart from existing foundation models in how we design our encoder g_ϕ, with an eye towards mitigating any dependency on troublesome pre-training data collection.

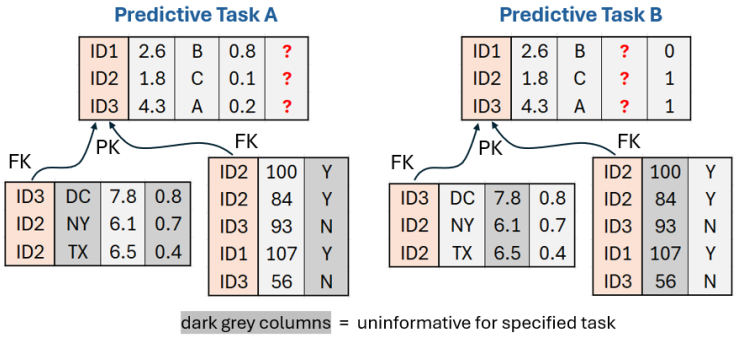

Figure 1: Example RDB (w/ $K = 3$ tables) where task-dependent column importance cannot be determined at the encoder stage. ICL samples are needed to resolve the intrinsic column ambiguity.

3.1 An RDB Encoder Specifically for ICL

Our guiding principle for RDB encoder design is as follows:

> There is no free lunch; RDB subgraphs may be large so any encoder **must** compress somewhere. Our insight is that at the encoder level, prior to seeing labeled ICL samples, we should only compress along the vertical dimension **within table columns**, where shared units facilitate interpretable aggregation guided by known PK-FK relations. Meanwhile, the encoder should **not** compress in the horizontal dimension **across table columns**, where roles, data types, and predictive relevance are broadly different and yet indeterminate without per-dataset/task label information. See Figure 1 for an illustration.

We formalize these considerations through a particular encoder definition targeting ICL application.

Definition 3.1. We define a *JUICE* encoder function g_{juice} (for "just use intra column encodings") if there exists a function $f_{\mathcal{R}}$, dependent on $\mathcal{R}$, such that the following hold:

1. $\boldsymbol{z}_{i:} \equiv \{z_{il}\}_{l=1}^{d_z} = g_{\text{juice}}[\mathcal{G}_H(\boldsymbol{x}_{i:})]$ for all rows of $\boldsymbol{X}$;
2. For each $l \in \{1, \ldots, d_z\}$ there is a $k \in \{1, \ldots, K\}$ and $j \in \{1, \ldots, d_k\}$ such that
$$\boldsymbol{z}_{:l} = f_{\mathcal{R}}\left(\boldsymbol{t}^k_{:j}\right) \quad \text{and} \quad \boldsymbol{z}_{:l} \perp\!\!\!\perp \mathcal{T} \backslash \boldsymbol{t}^k_{:j}. \tag{3}$$

Per this definition, if we convert each RDB subgraph to a fixed-length representation $\boldsymbol{z}_{i:}$ using JUICE, every column $\boldsymbol{z}_{:l}$ in the stacked representation matrix $\boldsymbol{Z}$ is a function of just *a single tabular data column from the original RDB*. Therefore each $\boldsymbol{z}_{:l}$ reflects the identity and functional role of a single RDB column, and no other. Arbitration of which columns are actually important for making predictions is then deferred to q_θ, where access to ICL samples facilitates informed decisions.

3.2 A 1D GNN Implementation

We now address practical ways to actually construct JUICE embeddings. The high-level idea to first split a given $\mathcal{G}_H(\boldsymbol{x}_{i:})$ into separate 1D subgraphs associated with each tabular column dimension, and then apply a specialized GNN to these subgraphs independently.

1D column-wise subgraph formation. For each tabular data column $\boldsymbol{t}^k_{:j}$ we form the revised subgraph $\mathcal{G}^{k,j}_H(\boldsymbol{x}_{i:})$, which is equivalent to $\mathcal{G}_H(\boldsymbol{x}_{i:})$ but with truncated node features dependent only on $\boldsymbol{t}^k_{:j}$. Specifically, for nodes extracted from any arbitrary table $\boldsymbol{T}^{k'}$, the original $d_{k'}$-dimensional row features are simply reduced to a 1D feature via

$$\boldsymbol{t}^{k'}_{i':} \rightarrow t^k_{i'j} \;\text{ if }\; k' = k, \quad \text{otherwise } \boldsymbol{t}^{k'}_{i':} \rightarrow 0, \quad \forall i'. \tag{4}$$

By design, any encoder applied to the revised subgraph $\mathcal{G}^{k,j}_H(\boldsymbol{x})$ will only depend on column $\boldsymbol{t}^k_{:j}$, *retaining independence from all other RDB columns* as desired. We describe a general procedure for such encoder construction next.

Meta-path GNN layers. Although we have simplified our subgraphs via (4), each $\mathcal{G}^{k,j}_H(\boldsymbol{x}_{i:})$ nonetheless retains rich heterogeneous relationships through different PK-FK pairs within $\mathcal{R}$. These pairs determine a set of multi-relational meta-paths (Ferrini et al., 2024) between a target node defined by a row $\boldsymbol{x}_{i:}$, and the nodes associated with rows of other tables within a H-hop radius.[1] Analogous to tabular data columns with varying relevance, prior to seeing ICL samples the encoder cannot possibly know which meta-paths are most discriminative for a given predictive task.

A typical heterogeneous GNN (Busbridge et al., 2019; Hu et al., 2020; Schlichtkrull et al., 2018) would naturally interleave all of these meta-paths together in forming predictions. But this can be counter-productive outside of the supervised learning setting, where back-propagated gradients reflecting supervision labels are available to selectively discount the less important meta-paths on a dataset-by-dataset basis. Fortunately though, we can instead encode *separate* 1D representations associated with each meta-path using a meta-path GNN. Subsequently q_θ equipped with ICL samples can determine relevance, as candidate meta-paths remain conveniently column-aligned in the augmented feature space.

Given a length-H meta-path ρ_H extracted from $\mathcal{G}^{k,j}_H(\boldsymbol{x}_{i:})$, we initialize all 1D input-layer node embeddings denoted $\boldsymbol{\mu}^{(0)}$ using (4). From there, embedding updates (Ferrini et al., 2024) for a node v on layer $h+1$ are given by

$$\mu^{(h+1)}_v = \sigma\left(w^{(h)}_0 \mu^{(h)}_v + \text{agg}\left[\left\{w^{(h)}\mu^{(h)}_u\right\}_{u\in\mathcal{N}^{(h)}_v}\right]\right), \tag{5}$$

where agg is a permutation invariant aggregation function (e.g., sum, mean) and σ is an activation (e.g., linear, ReLU, leaky-ReLU, etc.). Additionally, $\mathcal{N}^{(h)}_v$ denotes the set of neighbors of node v according to the relation or edge type associated with step h along H (Ferrini et al., 2024). We also assume meta-paths traverse each table at most once, before converting output layer embeddings to elements of $\boldsymbol{Z}$.

Resulting JUICE encoder design. Although not the most computationally efficient way to implement g_{juice}, we now summarize the constituent steps from a conceptual standpoint for transparency (full implementation details and complexity considerations will be addressed in Section 5):

1. Given a new RDB, select a row $\boldsymbol{x}_{i:}$ from $\boldsymbol{X} \equiv \boldsymbol{T}^K$ and initialize $\boldsymbol{z}_{i:}$ to $\boldsymbol{x}_{i:}$.
2. Extract $\mathcal{G}_H(\boldsymbol{x}_{i:})$ as in prior work on supervised RDB predictive models.

[1] A meta-path ρ_H is a sequence of H edges connecting properly-typed nodes in $\mathcal{G}^{k,j}_H(\boldsymbol{x}_{i:})$. For example, on an e-commerce platform we could have *user* $\xrightarrow{\text{purchased}}$ *product* $\xrightarrow{\text{sold\_by}}$ *seller*.

3. Choose a tabular data column $\boldsymbol{t}_{:j}^{k}$ from within an H-hop radius of $\boldsymbol{x}_{i:}$ over $\mathcal{G}_H(\boldsymbol{x}_{i:})$, and convert $\mathcal{G}_H(\boldsymbol{x}_{i:})$ to $\mathcal{G}_H^{k,j}(\boldsymbol{x}_{i:})$ using (4).
4. From within $\mathcal{G}_H^{k,j}(\boldsymbol{x}_{i:})$, select a meta-path ρ_H between the node associated with row $\boldsymbol{x}_{i:}$ and the rows in table k, and propagate node embeddings using (5).
5. Collect 1D node embeddings associated with $\boldsymbol{x}_{i:}$ from output layer and concatenate to $\boldsymbol{z}_{i:}$.
6. Repeat the above, looping over meta-paths and data columns within H-hops of $\boldsymbol{x}_{i:}$.
7. Output final embedding $\boldsymbol{z}_{i:} = g_{\text{juice}}[\mathcal{G}_H(\boldsymbol{x}_{i:})]$.

If we compute JUICE embeddings for all rows of $\boldsymbol{X}$ as described above, the resulting $\boldsymbol{Z}$ will necessarily satisfy Definition 3.1. In fact, the interpretable column-wise information partitioning is even stronger: Each column of $\boldsymbol{Z}$ so-constructed is actually a function of a single auxiliary data column *and* a single meta-path connecting the target node with nodes within this data column. This organization dramatically simplifies the ICL stage, which need not disentangle useful and useless dimensions that have been nonlinearly coupled through a traditional dense encoder.

Additional JUICE variations. There exist other channels for enhancing JUICE expressiveness as well. Specifically, we may execute (5) with multiple different agg functions and concatenate the results to obtain a richer set of column-aligned representations. Depending on the task, different aggregations may be useful in sorting out various forms of homophily versus heterophily network effects (see Appendix D for further background context and related discussion). For example, for an RDB predictive task dominated by homophily relationships, mean or mode aggregation could potentially be quite valuable. Meanwhile, stdev, entropy, or quantile aggregations could be suitable for capturing a portion of network effects leaning in the heterophily direction. Concatenation of multiple such aggregation functions has also been advocated in Corso et al. (2020), where it is shown that a single aggregator alone cannot differentiate various non-isomorphic graph structures.

3.3 Encoder Training is Not Necessary

For a generic GNN or Transformer-based encoder with multi-dimensional node features, trainable linear filters are critical for increasing performance. However, by design our situation only requires 1D node features, which facilitates helpful simplifications per the following:

Proposition 3.2. *If σ and agg are positively homogeneous functions, we initialize embeddings using (4), and $\{w_0^{(h)}, w^{(h)}\} \in \mathbb{R}_+ \ \forall h$, then without loss of generality (5) can be reparameterized with no internal weights.*

We remark that many of the most commonly used activations (e.g., ReLU, leaky-ReLU, linear) and aggregations (e.g., mean, sum, mode, min, max, stdev) are positively homogeneous, so this requirement is not a significant limitation. Additionally, while assuming positive internal weights does impose some modest form of constraint, sign information can be re-introduced by absorbing into revised σ and agg definitions. Multiple such variations can be incorporated into JUICE to recover full expressiveness (in Section 3 we also advocate for inclusion of multiple aggregators).

As such, by virtue of Proposition 3.2, we can remove parameters from our 1D GNN implementation of JUICE without any appreciable loss of expressiveness. Note that even if such a reparameterization produces additional scale factors on the output layer (as opposed to internal weights), these can be absorbed into q_θ. In fact, normalizing ICL feature dimensions (a common pre-processing tactic (Hollmann et al., 2025)) removes such scale factors anyway.

3.4 Combining JUICE with Single-Table ICL Model

By preserving column roles and identities (without conversion to traditional dense embeddings that fuse cross-column units and network effects), ICL samples $\{\boldsymbol{z}_{i:}, y_i\}_{i=1}^{n}$ produced by JUICE retain the canonical form for which single-table foundation models were originally designed. And critically, because this is achievable via a parameter-free encoding process, no additional RDB data collection and pre-training steps are required at any stage if we simply pair JUICE with one of these existing single-table models for instantiating the decoder q_θ, e.g., TabPFN (Hollmann et al., 2025).

It is worth emphasizing that models like TabPFN are highly capable of processing multiple disparate tabular feature columns, pruning away the unnecessary ones, and making predictions (Zhang et al.,

2025a); they are not especially sensitive to the specific marginal distributions of each column that may change with vertical aggregations. As such, by construction JUICE largely preserves current TabPFN advantages and can leverage future enhancements as well.

4 Analytical Support for JUICE

Prior to seeing ICL samples with labels, it is not possible for any dataset-agnostic encoder (meaning one that has not undergone supervised training on a specific dataset) to determine from individual subgraphs alone which constituent features are informative and which are not. This is because the role and utility of any given feature itself, such as a column in an auxiliary table or a meta-path connecting different entity types, can vary from RDB to RDB, and even from task to task within a single RDB. In this section we further examine this fundamental limitation of fixed encoders in two complementary respects as related to the ICL setting.

4.1 Limitations of Cross-Column Dense Embeddings

We first consider a simplified data generation scenario introduced to isolate the consequences of using compressed embeddings from an arbitrary dense encoder. Similar consequences emerge from more complex setups as well at the cost of presentation clarity.

Definition 4.1. We define $\mathcal{D}(\pi)$ as a dataset generated using a function $\pi : \mathbb{R}^\kappa \to \mathbb{R}$ as follows. First draw samples $\boldsymbol{x}_{i:} \in \mathbb{R}^d$ from some $p(\boldsymbol{x}_{i:})$, where $i = 1, \ldots, n+1$ and $p(\boldsymbol{x})$ is assumed to be an absolutely continuous distribution satisfying $p_{\min} \leq p(\boldsymbol{x}) \leq p_{\max}$ for all $\boldsymbol{x} \in \mathcal{X}^d$.[2] We next select a set of κ indices from $\{1, \ldots, d\}$ uniformly at random (without replacement) and form reduced features $\widetilde{\boldsymbol{x}}_{i:} \in \mathbb{R}^\kappa$. And finally, we compute labels $y_i = \pi(\widetilde{\boldsymbol{x}}_{i:})$ for all i. We then reserve $\{\boldsymbol{x}_{i:}, y_i\}_{i=1}^n$ for ICL while treating sample $n+1$ as $\{\boldsymbol{x}_{\text{test}}, y_{\text{test}}\}$.

By design of this generative process, uninformative features correspond with dimensions of $\boldsymbol{x}_{i:}$ that have been pruned in forming the reduced set $\widetilde{\boldsymbol{x}}_{i:}$, which alone is relevant to predicting y_i within any given dataset. In the context of *supervised training* involving a single dataset draw, it is natural to learn compressed encoder representations $\boldsymbol{z}_{i:} = g_\phi(\boldsymbol{x}_{i:}) \in \mathbb{R}^r$ with $\kappa \leq r < d$. For example, an ideal scenario would have this encoder simply learning to discard all unnecessary dimensions of the original features $\boldsymbol{x}_{i:}$, such that making optimal predictions with the resulting compressed representation $\boldsymbol{z}_{i:} \approx \widetilde{\boldsymbol{x}}_{i:}$ is substantially easier. As a stark contrast, such simplifying encoder compression is *not* generally possible when we turn to *ICL settings*:

Proposition 4.2. *Assume $\pi : \mathbb{R}^\kappa \to \mathbb{R}$ is affine, with weights in general position. Then for any $n > \kappa$, there exists a fixed ICL decoder[3] f_θ such that*

$$\left| y_{\textit{test}} - f_\theta\big(\{\boldsymbol{x}_{i:}, y_i\}_{i=1}^n, \boldsymbol{x}_{\textit{test}}\big) \right| = 0 \tag{6}$$

with probability one over datasets $\mathcal{D}(\pi)$ generated according to Definition 4.1. Meanwhile, for any $r < d$, and any possible fixed encoder-decoder pair $\{g_\phi, f'_\theta\}$, we have

$$\left| y_{\textit{test}} - f'_\theta\big(\{\boldsymbol{z}_{i:}, y_i\}_{i=1}^n, \boldsymbol{z}_{\textit{test}}\big) \right| > 0, \quad \boldsymbol{z} = g_\phi(\boldsymbol{x}) \in \mathbb{R}^r \tag{7}$$

with probability at least $1 - \binom{r}{k} / \binom{d}{k}$ over the same data generative distribution.

This result can be generalized to nonlinear data generation schemes and further elucidated with more precise error bounds. However, the core message of Proposition 4.2 is straightforward and can be conveyed without these extensions: *Ideal per-dataset compression of informative and uninformative feature columns is not possible with an encoder that must remain fixed across a distribution of input datasets, where column roles are subject to change*.

4.2 Sample Complexity Considerations

In Section 4.1 we assumed an encoder stage that formed dense, compressed representations with dimensionality $r < d$. We now turn to the case where $r = d$, i.e., no enforced compression across feature columns. In this regime we examine the degree to which a dense encoder g_θ may still negatively influence the sample complexity required to achieve a given estimation error.

[2] Each such generated $\boldsymbol{x}_{i:}$ can be viewed as a set of features collected from one or more tables.

[3] For simplicity in this section, we adopt a deterministic decoder f_θ, as opposed to the probabilistic version q_θ introduced earlier.

At first glance this possibility may seem counter-intuitive. After all, during the training phase encoder-decoder pairs $\{g_\phi, f_\theta\}$ can (in principle) learn to coordinate in such a way that a parameterized g_ϕ producing dense cross-column representations is helpful. This is certainly true for new inference tasks that lie well *within* the distribution of the pre-training data. However, most real-world applications of ICL will inevitably deviate from this (likely synthetic) distribution, and it is here that a complex, parameterized encoder (as distinct from the decoder) can introduce problems.

To better understand this phenomena, we note that single-table ICL via a *decoder alone* can be relatively stable to distribution shifts (Zhang et al., 2025a), and asymptotically consistent under the right conditions (Nagler, 2023). In principle then, we can still obtain reasonable results even for new datasets that may substantially differ in distribution from a synthetic training set. Now consider the addition of an encoder, which sees *only a single input feature vector at a time*, not the full set of ICL samples; an OOD input here can induce a substantially different encoder representation. And yet the encoder's behavior is only "known" to the decoder indirectly over the support of the pre-training data. And so from the decoder's standpoint, such a representation can lose any coordinated characteristics that might otherwise reduce estimation difficulty or sample complexity. In other words, in OOD regimes the encoder can essentially behave like an unknown transform, and if uninformative features exist within the original column-wise frame of reference, they will now be mixed by a process unknown to the decoder.

This difference can be dramatic. Working in the original feature frame leads to a sample complexity scaling with an exponential dependency on κ (the number of informative features), while the OOD encoder can push this rate to be much worse, scaling exponentially with d (the ambient dimension), even on the simplest of estimation problems. We quantify this phenomena as follows; see also Appendix B for empirical corroboration.

Proposition 4.3. *Let $\mathcal{F}$ denote the set of Lipschitz continuous functions on $[0,1]^\kappa$. Then there exists an ICL decoder f_θ such that (excluding smaller-order terms) we have*

$$\sup_{\pi \in \mathcal{F}} \mathbb{E}_{\mathcal{D}(\pi) \sim p} \Big[y_{\textit{test}} - f_\theta\big(\{\boldsymbol{x}_{i:}, y_i\}_{i=1}^n, \boldsymbol{x}_{\textit{test}}\big) \Big]^2 = O\left(n^{-2/\kappa}\right) \tag{8}$$

where the data distribution p is given by Definition 4.1 with $\mathcal{X} = [0,1]$. In contrast, with $\kappa = 1$ and $y = \widetilde{x}$, we have

$$\inf_{f_\theta} \sup_{g_\phi \in \mathcal{B}} \mathbb{E}_{\mathcal{D}(\pi) \sim p} \Big[y_{\textit{test}} - f_\theta\big(\{\boldsymbol{z}_{i:}, y_i\}_{i=1}^n, \boldsymbol{z}_{\textit{test}}\big) \Big]^2 = \Theta\left(n^{-2/d}\right) \tag{9}$$

where $\boldsymbol{z} = g_\phi(\boldsymbol{x})$ represents an encoder selected from the set of bi-Lipschitz bijections $\mathcal{B}$ mapping $[0,1]^d \to [0,1]^d$.

5 RDBLEARN: A LIGHTWEIGHT RDB TOOLKIT

We develop the open-source package *RDBLearn*[4] to operationalize the merger of JUICE with single-table foundation models as advocated in Section 3.4. The result is an easy-to-use RDB toolbox with no training required. See Zhang et al. (2026) for specifics of RDBLearn usage and design, including its optional agent-specific interface. We highlight several attributes here:

- *Intuitive interface*: The programming interface is designed to mirror the underlying formulation: core objects in the API correspond with concepts such as the relational database context $\{\mathcal{T}, \mathcal{R}\}$, ICL samples $\mathcal{D}$, and per-instance relational neighborhoods $\mathcal{G}_H(\boldsymbol{x})$. This makes it straightforward to relate experimental configurations and results back to modeling assumptions.
- *SQL-backed JUICE optimizations*: We implement JUICE principles using DFS primitives (Kanter & Veeramachaneni, 2015) through SQL execution over relational tables, leveraging the database engine for joins and aggregations. System optimizations include: (i) Translating feature synthesis primitives into SQL queries with aggregation pushdown; (ii) Reusing intermediate results through caching or incremental materialization; and (iii) Compiling cutoff-time constraints into the SQL execution plan to avoid temporal leakage when predicting future targets.
- *ICL-model agnostic design*: Any single-table ICL predictor that follows the scikit-learn estimator interface can be directly incorporated. This supports controlled comparisons and upgrades under the same relational pipeline.

[4] `https://github.com/HKUSHXLab/rdblearn`

- *Temporal OOD reduction*: While base ICL predictors like TabPFN have been applied to forecasting tasks (Hoo et al., 2024), this requires introducing an array of cyclic temporal features to avoid OOD effects from unseen time stamps in the inference window. We sidestep this issue by encoding relative temporal differences rather than absolute times when forming ICL samples.

6 Testing with Real-World RDBs

For all experiments, RDBLearn extracts JUICE embeddings and converts to ICL samples using official benchmark training splits. We then choose $H \in \{2, 3\}$ and the RDBLearn base model from TabPFNv2 (Hollmann et al., 2025), TabPFNv2.5 (Grinsztajn et al., 2025), and LimiX (Zhang et al., 2025b) using dev sets (although RDBLearn is stable across these choices as shown below). No other hyperparameters or tuning is involved. Appendix E contains additional experiment details.

		no supervised training →							
		LLM-A	LLM-B	RelLLM	Griffin	RT	KumoRFM	RDBLearn	RelGT
rel-amazon	item-churn	62.1	71.96	64.1	71.9	74.3	79.93	82.07	82.55
	user-churn	58.1	60.56	60.07	64.1	65.2	67.29	67.57	70.39
rel-avito	user-clicks	59.8	61.32	62.28	45.9	60.8	64.11	69.04	68.30
	user-visits	62.7	60.28	56.17	62.2	62.6	64.85	65.49	66.78
rel-hm	user-churn	59.8	64.34	55.95	60.4	63.1	67.71	68.05	69.27
rel-stack	user-badge	80.0	71.13	62.12	82.3	83.6	80.00	85.26	86.32
	user-engage	78.0	81.01	69.46	89.4	87.8	87.09	89.39	90.53
rel-trial	study-outcome	57.4	55.72	59.02	57.2	60.1	70.79	71.58	68.61
	mean	64.74	65.79	61.15	66.68	69.69	72.72	74.81	75.34

	H=2	*H*=3
TabPFN (v2)	73.94	74.55
TabPFN (v2.5)	72.03	74.30
LimiX	73.03	73.74

Figure 2: *Left*: Entity classification results (AUC) on RelBench; yellow is best, orange is second best among untrained models. *Right*: RDBLearn ablation (mean AUC on RelBench).

Baselines. We compare against the schema-agnostic **RT** (relational transformer) (Ranjan et al., 2025), **Griffin** (Wang et al., 2025), and **RelLLM** (Wu et al., 2025) models, all of which were pre-trained using the RelBench datasets (Robinson et al., 2024b). As for pure language model baselines, we report results from **LLM-A** (Team et al., 2025) (as tested in Ranjan et al. (2025)) and **LLM-B** (Wydmuch et al., 2024); both of these rely on serialized RDB neighborhood representations and/or ICL samples, and both can operate without any RDB pre-training or per-dataset fine-tuning on classification tasks. Although not a verifiable baseline, for reference we also include the closed-source **KumoRFM** industry model (Fey et al., 2025). And finally, as a representative contrast outside of the foundation model scope of the others, we consider **RelGT** (Dwivedi et al., 2025), a SOTA *fully-supervised* approach that shares the same dense encoder as KumoRFM. In this way, at a conceptual level the key distinction between KumoRFM and RDBLearn lies in the choice of encoder, with the latter alone based on JUICE foundations from Sections 3 and 4. All model hyperparameters were optimized per the specifications in prior work.

RelBench classification results. Figure 2(*left*) presents results on the RelBench datasets, excluding rel-event and rel-f1, both of which have label leakage concerns noticed by ourselves and others (see Appendix E for details). Overall, RDBLearn outperforms prior foundation models, and is the only RDB foundation model that is competitive with the fully supervised RelGT approach. This is notable given that RDBLearn is only exposed to synthetic single-table data (during pre-training of the tabular base model); in contrast RelLLM, RT, and Griffin (and possibly others) have been exposed to each of the actual benchmarks as part of the pre-training adopted to produce the results in Figure 2. See Appendix A for further discussion of these models w.r.t. zero-shot learning.

RDBLearn stability. Figure 2(*right*) demonstrates the stability of RDBLearn as the base tabular prediction model and encoder hops H are varied on the same RelBench tasks. Notably, for every combination except one the performance exceeds all other RDB foundation models from Figure 2.

RelBench regression results. Regression poses a unique set of challenges to RDB foundation models, and prior work often concedes that regression results are unsatisfactory without per-dataset fine-tuning, particularly for LLM-based approaches (Wu et al., 2025; Wydmuch et al., 2024). For this reason we have fewer baselines to compare against, which are further compromised by inconsistent metrics and reproducibility issues. Hence Figure 3(*left*) compares only against KumoRFM (and RelGT for reference), as no other foundation models are directly comparable (see Appendix

E for discussion of disanalogous Griffin and RT results). We remark that RDBLearn matches KumoRFM performance, and is even competitive with supervised RelGT, despite no regression-specific modifications or adjustments. It is unknown what regression allowances and/or regression-specific real-world pretraining data have been incorporated into KumoRFM.

		KumoRFM	RDBLearn	RelGT
rel-amazon	item-ltv	55.254	48.559	48.922
	user-ltv	16.161	14.540	14.267
rel-avito	ad-ctr	0.035	0.034	0.035
rel-hm	item-sales	0.040	0.064	0.054
rel-stack	post-votes	0.065	0.068	0.065
rel-trial	site-success	0.417	0.424	0.326
	study-adverse	58.231	43.913	43.992
	normalized mean	1.088	1.074	1.000

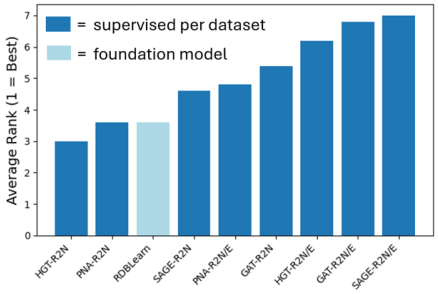

Figure 3: *Left*: Regression results (MAE) on RelBench. *Right*: Classification ranking on 4DBInfer.

Additional 4DBInfer comparisons. Prior work on RDB foundation models has focused on RelBench tasks as above. However, to further establish the native versatility of RDBLearn, we apply our identical pipeline (again with zero modification whatsoever besides coupling with a suitable data loader) to the classification tasks drawn from the 4DBInfer benchmark (Wang et al., 2024). We compare against a suite of heterogeneous GNN and graph Transformer models specifically adapted for this benchmark with multiple graph extraction techniques; each such approach also benefits from per-dataset supervised learning and extensive hyperparameter optimization (see Appendix E). Results are shown in Figure 3(*right*), where RDBLearn exhibits strong performance without training, leading to orders of magnitude greater efficiency (see Appendix E).

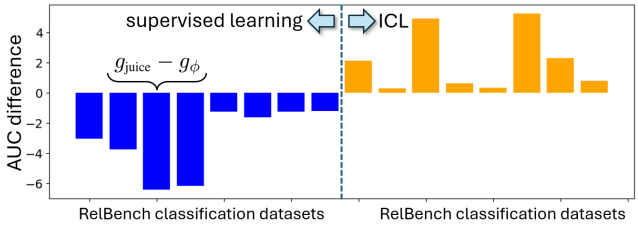

Figure 4: Performance inversion of JUICE embeddings.

JUICE is worth the squeeze. If the principles underpinning JUICE (as used by RDBLearn) are sound, then we should expect a notable gap in performance when we contrast supervised learning versus ICL usage. Specifically, in the SL regime we should expect that an expressive parameterized g_ϕ dominates g_{juice}, while the situation should largely flip when we turn to ICL cases. To empirically explore this phenomena, which has *not* been previously recognized, we consider the KumoRFM ICL model and the supervised RelGT model, both of which share the same g_ϕ. Meanwhile, for g_{juice} we can pair with both an ICL decoder (as within RDBLearn) and a strong supervised prediction head such as AutoGluon (Erickson et al., 2020). Given these four model instantiations, we plot the corresponding $g_{\text{juice}} - g_\phi$ performance differences split across SL and ICL cases in Figure 4. The outcome closely conforms with the expected performance inversion. In Appendix C we repeat an analogous experiment drawing on data from prior ICL-based graph foundation model studies; the outcome is similar. These results demonstrate the wider relevance of treating JUICE as an end goal, and not merely a simplifying approximation.

7 Conclusions

For RDB supervised learning, conventional wisdom points towards expressive parameterized encoders for converting variably-sized subgraph information into dense discriminative representations for end-to-end training. Meanwhile more traditional parameter-free RDB featurization steps are often viewed as outdated heuristics. However, we have argued that these designations need not hold as we move to ICL-based RDB foundation models. We ultimately exploit these findings through our RDBLearn toolbox, obviating the need for any complex pre-training or RDB data collection.

REFERENCES

Maya Bechler-Speicher, Ben Finkelshtein, Fabrizio Frasca, Luis Müller, Jan Tönshoff, Antoine Siraudin, Viktor Zaverkin, Michael M Bronstein, Mathias Niepert, Bryan Perozzi, et al. Position: Graph learning will lose relevance due to poor benchmarks. *arXiv preprint arXiv:2502.14546*, 2025.

Gérard Biau and Luc Devroye. *Lectures on the nearest neighbor method*, volume 246. Springer, 2015.

Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y Hammerla. Relational graph attention networks. *arXiv preprint arXiv:1904.05811*, 2019.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Zhikai Chen, Han Xie, Jian Zhang, Jiliang Tang, Huzefa Rangwala, George Karypis, et al. AutoG: Towards automatic graph construction from tabular data. *arXiv preprint arXiv:2501.15282*, 2025.

Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. ARDA: Automatic relational data augmentation for machine learning. *Proc. VLDB Endow.*, 13(9):1373–1387, May 2020.

Dongwon Choi, Sunwoo Kim, Juyeon Kim, Kyungho Kim, Geon Lee, Shinhwan Kang, Myunghwan Kim, and Kijung Shin. RDB2G-Bench: A comprehensive benchmark for automatic graph modeling of relational databases. *arXiv preprint arXiv:2506.01360*, 2025a.

Jeongwhan Choi, Woosung Kang, Minseo Kim, Jongwoo Kim, and Noseong Park. Can TabPFN compete with GNNs for node classification via graph tabularization? *arXiv preprint arXiv:2512.08798*, 2025b.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

Milan Cvitkovic. Supervised learning on relational databases with graph neural networks. *arXiv preprint arXiv:2002.02046*, 2020.

DM Dave, B Todd, and Will Cukierski. Acquire valued shoppers challenge, 2014. URL `https://kaggle.com/competitions/acquire-valued-shoppers-challenge`.

D.L. Donoho and M. Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proc. National Academy of Sciences*, 100(5), 2003.

Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer. *arXiv preprint arXiv:2505.10960*, 2025.

Dmitry Eremeev, Gleb Bazhenov, Oleg Platonov, Artem Babenko, and Liudmila Prokhorenkova. Turning tabular foundation models into graph foundation models. *arXiv preprint arXiv:2508.20906*, 2025a.

Dmitry Eremeev, Oleg Platonov, Gleb Bazhenov, Artem Babenko, and Liudmila Prokhorenkova. GraphPFN: A prior-data fitted graph foundation model. *arXiv preprint arXiv:2509.21489*, 2025b.

Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505*, 2020.

Francesco Ferrini, Antonio Longa, Andrea Passerini, and Manfred Jaeger. Meta-path learning for multi-relational graph neural networks. In *Learning on Graphs Conference*, pp. 2–1. PMLR, 2024.

Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Relational deep learning: Graph representation learning on relational databases. *arXiv preprint arXiv:2312.04615*, 2023.

Matthias Fey, Vid Kocijan, Federico Lopez, J Lenssen, and Jure Leskovec. KumoRFM: A foundation model for in-context learning on relational data. *Online Tech Report*, 2025.

Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. SIGN: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

Quan Gan, Minjie Wang, David Wipf, and Christos Faloutsos. Graph machine learning meets multi-table relational data. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 6502–6512, 2024.

Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall, 2009.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. *Advances in Neural Information Processing Systems*, 34:18932–18943, 2021.

Léo Grinsztajn, Klemens Flöge, Oscar Key, Felix Birkel, Philipp Jund, Brendan Roof, Benjamin Jäger, Dominik Safaric, Simone Alessi, Adrian Hayler, et al. Tabpfn-2.5: Advancing the state of the art in tabular foundation models. *arXiv preprint arXiv:2511.08667*, 2025.

Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. Deepfm: a factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*, 2017.

Bruce Hajek and Maxim Raginsky. ECE 543: Statistical learning theory. *Department of Electrical and Computer Engineering and the Coordinated Science Laboratory, University of Illinois at Urbana-Champaign*, 2021.

Adrian Hayler, Xingyue Huang, Ismail Ilkan Ceylan, Michael Bronstein, and Ben Finkelshtein. Bringing graphs to the table: Zero-shot node classification via tabular foundation models. *arXiv preprint arXiv:2509.07143*, 2025.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*, 2022.

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.

Shi Bin Hoo, Samuel Müller, David Salinas, and Frank Hutter. The tabular foundation model tabpfn outperforms specialized time series forecasting models based on simple features. In *NeurIPS Workshop on Time Series in the Age of Large Models*, 2024.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*, pp. 2704–2710, 2020.

QU Jingang, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *International Conference on Machine Learning*, 2025.

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 (DSAA)*, pp. 1–10. IEEE, 2015.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

Stefan Kramer and C Bessiere. A brief history of learning symbolic higher-level representations from data (and a curious look forward). In *IJCAI*, pp. 4868–4876, 2020.

Stefan Kramer, Nada Lavrač, and Peter Flach. *Propositionalization Approaches to Relational Data Mining*, pp. 262–291. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001. doi: 10.1007/978-3-662-04599-2_11.

Arun Kumar, Jeffrey Naughton, Jignesh M Patel, and Xiaojin Zhu. To join or not to join? thinking twice about joins before feature selection. In *Proceedings of the 2016 International Conference on Management of Data*, pp. 19–34, 2016.

Jiabin Liu, Chengliang Chai, Yuyu Luo, Yin Lou, Jianhua Feng, and Nan Tang. Feature augmentation with reinforcement learning. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, pp. 3360–3372. IEEE, 2022.

mjkistler, Ran Locar, Ronny Lempel, RoySassonOB, Rwagner, and Will Cukierski. Outbrain click prediction, 2016. URL `https://kaggle.com/competitions/outbrain-click-prediction`.

Jan Motl and Oliver Schulte. The ctu prague relational learning repository. *arXiv preprint arXiv:1511.03086*, 2015.

Thomas Nagler. Statistical foundations of prior-data fitted networks. In *International Conference on Machine Learning*, pp. 25660–25676. PMLR, 2023.

Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 188–197, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1018. URL `https://aclanthology.org/D19-1018`.

Jakub Peleška and Gustav Šír. Tabular transformers meet relational databases. *ACM Transactions on Intelligent Systems and Technology*, 16(5):1–24, 2025.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: Unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational transformer: Toward zero-shot foundation models for relational data. *arXiv preprint arXiv:2510.06377*, 2025.

Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan Eric Lenssen, Yiwen Yuan, Zecheng Zhang, et al. Relational deep learning: Graph representation learning on relational databases. In *NeurIPS 2024 third table representation learning workshop*, 2024a.

Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan Eric Lenssen, Yiwen Yuan, Zecheng Zhang, et al. RelBench: A benchmark for deep learning on relational databases. *Advances in Neural Information Processing Systems*, 37:21330–21341, 2024b.

Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European semantic web conference*, pp. 593–607. Springer, 2018.

Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342*, 2021.

Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 Technical Report. *arXiv preprint arXiv:2503.19786*, 2025.

Minjie Wang, Quan Gan, David Wipf, Zheng Zhang, Christos Faloutsos, Weinan Zhang, Muhan Zhang, Zhenkun Cai, Jiahang Li, Zunyao Mao, et al. 4DBInfer: A 4D benchmarking toolbox for graph-centric predictive modeling on RDBs. *Advances in Neural Information Processing Systems*, 37:27236–27273, 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 *International Conference on Machine Learning*, 2025.

David Wipf and Bhaskar Rao. Probabilistic analysis for basis selection via ℓ_p diversity measures. *International Conference on Acoustics, Speech, and Signal Processing*, 6, May 2004.

Fang Wu, Vijay Prakash Dwivedi, and Jure Leskovec. Large language models are good relational learners. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025.

Marek Wydmuch, Łukasz Borchmann, and Filip Graliński. Tackling prediction tasks in relational databases with LLMs. *arXiv preprint arXiv:2411.11829*, 2024.

Lianghao Xia and Chao Huang. Anygraph: Graph foundation model in the wild. *arXiv preprint arXiv:2408.10700*, 2024.

Lianghao Xia, Ben Kao, and Chao Huang. Opengraph: Towards open graph foundation models. *arXiv preprint arXiv:2403.01121*, 2024.

Jaemin Yoo, Meng-Chieh Lee, Shubhranshu Shekhar, and Christos Faloutsos. Less is more: SlimG for accurate, robust, and interpretable graph mining. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3128–3139, 2023.

Lukáš Zahradník, Jan Neumann, and Gustav Šír. A deep learning blueprint for relational databases. In *NeurIPS 2023 Second Table Representation Learning Workshop*, 2023.

Han Zhang, Quan Gan, David Wipf, and Weinan Zhang. GFS: Graph-based feature synthesis for prediction over relational databases. *arXiv preprint arXiv:2312.02037*, 2023a.

Qiong Zhang, Yan Shuo Tan, Qinglong Tian, and Pengfei Li. TabPFN: One model to rule them all? *arXiv preprint arXiv:2505.20003*, 2025a.

Xingxuan Zhang, Gang Ren, Han Yu, Hao Yuan, Hui Wang, Jiansheng Li, Jiayun Wu, Lang Mo, Li Mao, Mingchao Hao, et al. Limix: Unleashing structured-data modeling capability for generalist intelligence. *arXiv preprint arXiv:2509.03505*, 2025b.

Xiyuan Zhang, Danielle C Maddix, Junming Yin, Nick Erickson, Abdul Fatir Ansari, Boran Han, Shuai Zhang, Leman Akoglu, Christos Faloutsos, Michael W Mahoney, et al. Mitra: Mixed synthetic priors for enhancing tabular foundation models. *arXiv preprint arXiv:2510.21204*, 2025c.

Yanlin Zhang, Linjie Xu, Quan Gan, David Wipf, and Minjie Wang. RDBLearn: Simple in-context prediction over relational databases. *arXiv preprint*, 2026.

Zizhao Zhang, Yi Yang, Lutong Zou, He Wen, Tao Feng, and Jiaxuan You. Rdbench: Ml benchmark for relational databases. *arXiv preprint arXiv:2310.16837*, 2023b.

Roman Zykov, Noskov Artem, and Anokhin Alexander. Retailrocket recommender system dataset, 2022. URL `https://www.kaggle.com/dsv/4471234`.

A Extended Related Work

Although we have already cited key references needed to understand and contextualize our contributions in the main text, there are several additional points worth bolstering here along with supporting references.

Growing relevance of RDB predictive modeling. As of 2026, the majority of database management systems are relational,[5] and the information stored within contains countless possibilities for predictive modeling. Traditionally, developing such models was predicated on some form of propositionalization (Chepurko et al., 2020; Kanter & Veeramachaneni, 2015; Kramer et al., 2001; Kramer & Bessiere, 2020; Kumar et al., 2016; Liu et al., 2022), whereby fixed-length features are first extracted and aggregated within a single table such that standard tabular learning models can then be applied. The latter range from diverse boosting approaches (Chen & Guestrin, 2016; Ke et al., 2017; Prokhorenkova et al., 2018) to deep learning frameworks like DeepFM (Guo et al., 2017), SAINT (Somepalli et al., 2021), and FT-Transformers (Gorishniy et al., 2021).

More recently, with the advent of graph neural networks and wide-ranging graph Transformer models, there has been a notable shift towards end-to-end systems applied to graphs extracted from RDBs (Cvitkovic, 2020; Dwivedi et al., 2025; Fey et al., 2023; Peleška & Šír, 2025; Wang et al., 2024; Zahradník et al., 2023; Zhang et al., 2023a). The underlying graph extraction process has also been the subject of ongoing exploration specifically for improving RDB prediction quality (Chen et al., 2025; Choi et al., 2025a; Gan et al., 2024). Of particular note, it has even been posited that the relevance of graph learning as a research domain in and of itself will fade unless focus is redirected towards (among other things) data originating in RDBs (Bechler-Speicher et al., 2025). For now though, evidence collected thus far suggests that in supervised settings, these recent end-to-end relational frameworks generally tend to outperform propositionalization, and conventional wisdom treats the latter as a bottleneck to be avoided where possible (Dwivedi et al., 2025). Of course as we have shown both analytically and empirically, this need not still be the case when we turn to foundation models like RDBLearn formulated through ICL. See also Appendix C where we broaden the scope of these observations to graph foundation models.

ICL and synthetic pre-training for RDB learning. For pure ICL-based RDB foundation models based on synthetic generation during pre-training, there is presently only the KumoRFM approach (Fey et al., 2025). However, as a closed-source model there are few details available that might otherwise enable thorough assessment by the research community. For example, while pre-training was achieved using a mixture of synthetic and real-world RDB data, it is unclear to what extent the real-world portion maintains structural or task similarity with the limited evaluation benchmarks. This calls into question how generalizable KumoRFM actually is in practice. Nor is the synthetic generation pipeline used by KumoRFM available for scrutiny that might inform its potential for widespread efficacy. We remark that generating synthetic RDBs that reflect real-world properties is challenging, with unavoidable dependency on extra relational dimensions of variability not shared by successful single-table models (Gan et al., 2024).

Zero-shot possibilities. Mirroring ambiguity shared across the graph learning literature (Eremeev et al., 2025a; Xia & Huang, 2024; Xia et al., 2024), there does not appear to be a widely agreed upon definition of what constitutes true zero-shot RDB predictive modeling. Broadly speaking though, zero-shot learning refers to settings whereby the model must predict test samples involving classes that were not available during training. This is possible in situations where there exists suitable auxiliary information (e.g., textual attribute descriptions) that implicitly differentiate new classes.

In the context of RDBs specifically, the relational transformer (RT) model has been framed as possessing zero-shot capabilities, provided zero-shot relational learning is defined as "predicting new targets on a new RDB with a new schema, without weight updates" as proposed by Ranjan et al. (2025). Per this definition our RDBLearn framework would also technically qualify as zero-shot. But in such relational cases the auxiliary information relied upon for making predictions (by both RT and RDBLearn) includes exposure to entity labels with earlier time-stamps, e.g., from the entity we wish to classify and/or those extracted from prescribed neighborhood(s). This setup is conceptually

[5] `https://db-engines.com/en/ranking_categories`

a bit like label propagation on a temporal graph, which is not universally recognized as zero-shot per se, although admittedly it is reasonable to consider broader definitions as long as stipulations are clear.

Regardless of definitions, if RT pre-training is not explicitly conducted using each test RDB, the performance drops appreciably, even approaching a naive baseline whereby the historical entity mean serves as the prediction. For example, on RelBench classification tasks, this historical mean estimator achieves 66.7 average AUC, while comparable RT performance is 70.1 AUC; see Table 1 in Ranjan et al. (2025). In contrast, if we strictly enforce no target labels (past or present) available at inference time as a requirement for zero-shot relational learning, then the Griffin model (Wang et al., 2025) still technically qualifies, although performance without per-dataset fine-tuning is not competitive (64 average AUC under equivalent settings).

Finally, the RelLLM model (Wu et al., 2025) has also been described as a zero-shot learner when per-dataset fine-tuning is omitted. However, RelLLM is still pre-trained over the very same RelBench datasets upon which testing is conducted (even if the pre-training targets may vary). Moreover, without per-dataset fine-tuning, RelLLM falls behind even Griffin (63.2 average AUC). We reiterate though, outside of the definition proposed by Ranjan et al. (2025), our RDBLearn would generally not be considered a pure zero-shot method given its reliance on ICL samples. Even so, unlike these other approaches, RDBLearn does *not* depend on seeing the actual inference-time RDBs during pre-training to achieve SOTA performance. Rather, it is entirely based on a synthetic pre-training pipeline whereby there is no possible leakage or favorable bias introduced from inference-time RDBs.

B Empirical Corroboration of Dense Encoder Performance Degradation

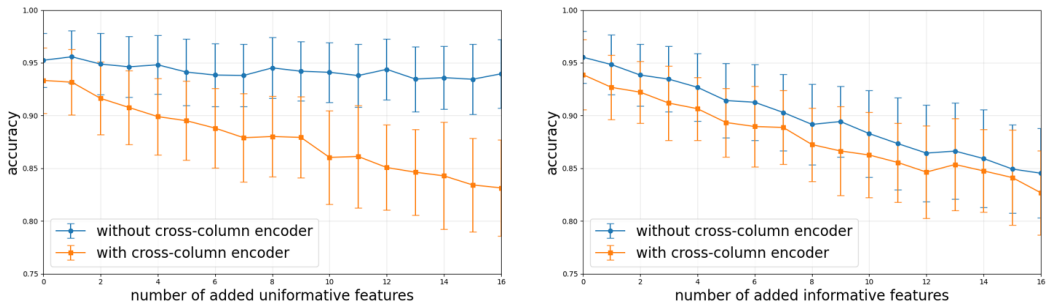

Figure 5: Impact of adding uninformative (*left*) versus informative (*right*) feature columns; results averaged over 100 trials.

To further explore our analytical findings from Section 4, we conducted the following simulation study involving two distinct testing scenarios. For the first, we initially generate ICL samples $\{\boldsymbol{X}, \boldsymbol{y}\} = \{\boldsymbol{x}_{i:}, y_i\}_{i=1}^n$, where elements of $\boldsymbol{X}$ are drawn iid from $\mathcal{N}(0,1)$ and $y_i = \mathbb{I}\big[1/\big(1 + \exp[-\boldsymbol{w}^\top \boldsymbol{x}_{i:}]\big)\big]$ is a binary class label. The weight vector $\boldsymbol{w}$ is also drawn iid from $\mathcal{N}(0,1)$. For all i we then compute representations $\boldsymbol{z}_{i:} = g_\phi(\boldsymbol{x}_{i:})$ that mix cross-column information (unlike JUICE). This encoder is composed of a randomized linear filter, followed by a leaky-ReLU nonlinearity, and another random linear filter. By design this encoder representation is invertible so no information is lost.

We next compute the ICL-based prediction accuracy of a decoder f_θ (implemented here via TabPFNv2) at new test points $\{\boldsymbol{x}_{\text{test}}, y_{\text{test}}\}$ as additional uninformative columns are randomly inserted into $\boldsymbol{X}$ and the corresponding elements of $\boldsymbol{x}_{\text{test}}$. We repeat the same procedure using ICL samples $\{\boldsymbol{Z}, \boldsymbol{y}\}$ evaluated at corresponding test points $\{\boldsymbol{z}_{\text{test}}, y_{\text{test}}\}$. Figure 5(*left*) displays the results (averaged over 100 independently generated datasets for each plotted point) with and without the addition of encoder g_ϕ. Of particular note, when no uninformative features are added (i.e., where the x-axis is zero on the left-hand plot), the encoder mixing has no appreciable effect; however, as

more distracting columns are added, a clear negative trend emerges as expected per the analysis of Section 4.

Meanwhile, our second testing scenario operates as a form of control. Specifically, instead of adding uninformative features to $\boldsymbol{X}$ as before, we now introduce additional columns upon which a revised $\boldsymbol{y}$ computation now explicitly depends (this is accomplished by extending the length of $\boldsymbol{w}$ during generation). In this regime, the core prediction problem naturally becomes harder, since the decision function becomes increasingly complex and high-dimensional. But critically, the inclusion of the cross-column encoder now has little effect as shown in Figure 5(*right*). This is because the encoder is no longer mixing informative and non-informative features that would otherwise complicate the predictive task faced by f_θ.

C Lessons from Graph Foundation Models Suggest that JUICE Is Worth the Squeeze

In Figure 4 from Section 6 we empirically demonstrated how the relative performance between a dense parameterized RDB encoder g_ϕ and our g_{juice} dramatically shifts when moving from supervised learning to ICL settings. Ideally we would like to extend these results via testing over additional RDB foundation models beyond KumoRFM, upon which Figure 4 is based. However, because no other comparable frameworks exist, it is unfortunately not possible to do so without developing our own completely new synthetic generation and pre-training pipeline specific to RDBs. Of course it remains an open question how to actually implement these modules, and well beyond the scope of this paper (recall that KumoRFM is a closed-source industry model with an undisclosed pre-training process).

Fortunately though, there does exist work on *graph foundation models* that we can leverage to bootstrap an analogous comparison, allowing us to further establish that the SL-to-ICL performance inversion predicted by our analysis represents a wide-ranging phenomena. Although not a perfect surrogate, data from homogeneous attributed graphs can be viewed as isomorphic to simplified RDBs involving a single data table type.

To this end, we now introduce a novel re-combination of performance results from the GraphPFN model (Eremeev et al., 2025b), which relies on a dense parameterized encoder g_ϕ to compress neighborhood subgraph information, and the G2T approach (Eremeev et al., 2025a), which uses a parameter-free encoder with similarities to JUICE. We loosely refer to the latter as $g_{\widehat{\text{juice}}}$, as neighboring node features are aggregated in a column-wise fashion; likewise for additional structured features such as node degree and PageRank score.

What is particularly attractive about this setup is that we have access to performance results whereby the core ICL decoder is shared across all models. This restricts variation along the two dimensions we wish to probe, namely, the $g_{\widehat{\text{juice}}} - g_\phi$ performance (difference) under supervised learning, and $g_{\widehat{\text{juice}}} - g_\phi$ under ICL. This is possible because the GraphPFN model is pre-trained from a LimiX check-point (Zhang et al., 2025b), and results from Tables 2 and 3 in Eremeev et al. (2025b) cover both ICL and fine-tuning (i.e., per-dataset supervision) cases. Meanwhile, these same tables include a LimiX version of G2T under the same conditions. Hence we pull these results and compute the respective differences stratified over each dataset and learning type as shown in Figure 6 (the number of supervised learning datasets is fewer since some cases ran OOM).

Just as in Figure 4, which involves completely different datasets and models, Figure 6 reveals the same clear performance inversion predicted for JUICE-based embeddings. This consistency helps solidify our conclusions and widens the applicability of underlying JUICE design principles, suggesting that even in the realm of graph foundation models, dense parameterized encoders (and attendant pre-training data collection) need not be necessary, at least outside of fine-tuning or supervised training regimes. We also remark that graph foundation models relying on parameter-free graph features served to single-table foundation models (like G2T) have been proposed in multiple concurrent works (Choi et al., 2025b; Eremeev et al., 2025a; Hayler et al., 2025). However, in all of these cases the design is motivated by simplicity and guided by natural precursors from supervised graph learning (Frasca et al., 2020; Yoo et al., 2023). Unlike herein, prior work does *not* establish formal principles for grounding such approaches, nor explicit elucidation of the critical distinction between supervised and ICL usage thereof.

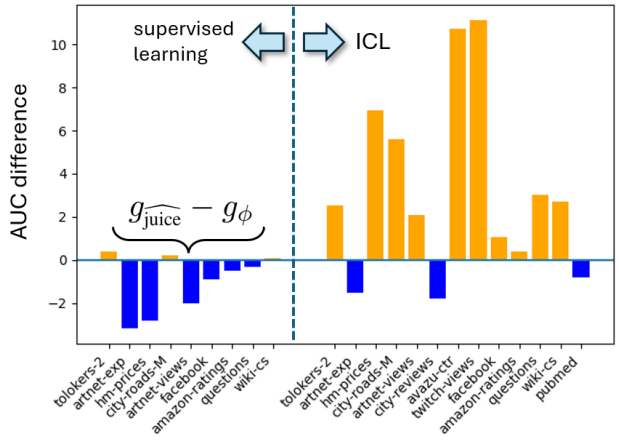

Figure 6: Performance inversion of JUICE-like embeddings on homogeneous graph datasets. Each bar plot is based on results extracted from Eremeev et al. (2025b) involving GraphPFN and G2T models formed with a LimiX ICL decoding architecture.

D Arbitrating Homophily vs Heterophily Relationships

Analogous to the feature- and meta-path-level ambiguities already discussed in the main text, there exists additional sources of uncertainty with respect to how network effects manifest in RDB predictive tasks. Depending on the task and dataset, unknown labels may be influenced by either heterophily or homophily relationships between neighboring nodes captured by distinct meta-paths. In the context of heterogeneous graphs defining RDB relations, homophily refers to the tendency of neighboring nodes of the same node type (not necessarily 1-hop neighbors) sharing the same label, while heterophily represents the converse. Differentiating such network effects is possible during supervised training on a per-dataset basis as gradients back-propagate through q_θ to g_ϕ. In this way it is possible in principle for a single architecture, albeit with dataset-specific weights, to handle varying degrees of homophily on a case by case basis. But the ICL setting is completely different. When no per-dataset label-aware gradients flow to g_ϕ, the encoder is blind to the importance of distinct meta-paths or the extent to which homophily vs heterophily dominates, and therefore is incapable of selectively preserving features that favor one effect over another. Hence suitable representations for *both* scenarios are needed prior to the ICL decoder. The JUICE philosophy suggests that these should be confined to distinct columns to the extent possible, such that subsequent ICL-based decoding is not unduly complex. Exploring this topic further represents an interesting direction for future research.

E Experiment Details

E.1 RDBLearn Setup

Encoding settings. For instantiating JUICE, we fix the activation function σ to be linear and choose agg functions $\{\text{sum}, \text{mean}, \text{mode}, \min, \max, \text{std}\}$ for continuous-valued columns and $\{\text{counts}, \text{mode}\}$ for categorical columns. These common/intuitive selections remain unchanged across all benchmarks, datasets, and tasks reported herein. We explicitly enable date/time features encoded as temporal differences to reduce potential OOD effects. However, we have chosen to simply disable all raw text, special text, and n-gram features. Interestingly, RDBLearn still achieves SOTA foundation model performance without these, indicating that textual attributions may not be central to making good predictions on current benchmarks (certainly this is often true of tabular data more broadly). In the future though, we can of course easily include a pre-trained language encoder to featurize text analogous to any other column type to further boost prediction quality.

ICL Decoder settings. We evaluate three foundation model variants for the RDBLearn ICL-based decoding step:

1. **TabPFNv2** (checkpoint: tabpfn-v2-classification/regression-finetuned-zk73skhh);
2. **TabPFN v2.5** (checkpoint: tabpfn-v2.5-classification/regression-default);
3. **Limix** (checkpoint: LimiX-16M).

For consistency over all benchmarks we randomly down-sample training splits to 10k. Although some ICL base models can handle up to 50k samples, and this limit is regularly increasing, we found that 10k was sufficient for good performance.

Equipment. All experiments were conducted on a single NVIDIA 4080 GPU with 32GB memory.

E.2 RELBENCH TESTING

We follow the official temporal splits (train, dev, test) from Robinson et al. (2024b) to facilitate direct comparison with existing published work. Moreover, results presented herein represent tuned models as reported by original authors with one exception. Griffin model results in Figure 2 were obtained from Ranjan et al. (2025), where head-to-head alignment with the relational transformer was conducted under so-called zero-shot settings (see Appendix A for further discussion of zero-shot definitions). The Griffin paper itself (Wang et al., 2025) does not include this type of experimentation, focusing more on optimizing performance through per-dataset fine-tuning.

Benchmark leakage issues. As part of RelBench (Robinson et al., 2024b), the rel-event dataset has been found to have temporal leakage issues (Ranjan et al., 2025). There is also reason to believe that rel-f1 may be compromised as well, given that fine-tuned models are capable of essentially perfect accuracy (99.61 AUC) despite these data being tied to sporting events (F1 racing) with non-negligible degrees of uncertainty over temporal splits. For these reasons, and following related studies elsewhere, we omit rel-event and rel-f1 from our empirical comparisons.

Regression reproducibility. Although promising RelBench regression results are reported in Ranjan et al. (2025) for a reduced set of models, the MAE metric is not used (e.g., as was used previously by KumoRFM). Moreover, for most datasets we were not able to reproduce the reported R^2 metric results even for the simple baseline "entity mean" estimator (and at the time of this writing, public code is not available for doing so).[6] The two exceptions are the rel-trial dataset tasks, namely, site-success and study-adverse. On these two datasets RDBLearn outperforms both Griffin and RT as shown in Figure 7.

	Griffin	RT	RDBLearn
site-success	2.6	5.2	5.5
study-adverse	-2.5	3.4	20.0

Figure 7: Regression results (R^2 scores, higher is better) on rel-trial tasks from RelBench.

E.3 4DBINFER TESTING

Following Wang et al. (2024), we adopt baselines formed from widely-used heterogeneous GNN architectures, including **R-SAGE** or **R-GCN** (Schlichtkrull et al., 2018), **R-GAT** (Busbridge et al., 2019), **HGT**, (Hu et al., 2020), and **R-PNA** (Corso et al., 2020). Each model type is then independently paired with each of two graph extraction techniques. The first is **R2N** (row-to-node), initially introduced by Cvitkovic (2020) and discussed in Section 2.1. The second is **R2N/E** (row-to-node/edge) as detailed in Gan et al. (2024). The importance of exploring multiple graphs is now well-established (Choi et al., 2025a). Collectively, this results in a total of 8 baselines as listed in Figure 3(*right*). Other more traditional baselines reported in Wang et al. (2024) have generally worse accuracy than these.

[6]We remark that the entity mean represents a critical signal for the RT model as shown in ablations from Ranjan et al. (2025).

For specific datasets, we focus on 5 of the 4DBInfer classification tasks. These include click-through-rate prediction on Outbrain (mjkistler et al., 2016), user churn on Amazon Book Reviews (Ni et al., 2019), user churn and post popularity prediction on StackExchange,[7] and conversion prediction on RetailRocket (Zykov et al., 2022). For reference, the raw results across all baselines and tasks are shown in Figure 8.

Dataset	Task	GAT(R2N)	GAT(R2N/E)	HGT(R2N)	HGT(R2N/E)	PNA(R2N)	PNA(R2N/E)	SAGE(R2N)	SAGE(R2N/E)	RDBLearn
Amazon	Churn (AUC ↑)	0.7622	0.7192	0.773	0.6864	0.7645	0.7157	0.7571	0.7314	0.7741
Outbrain	CTR-100K (AUC ↑)	0.6146	0.6308	0.626	0.6323	0.6249	0.6322	0.6239	0.6271	0.5447
Retailrocket	CVR (AUC ↑)	0.8284	0.7536	0.8495	0.8342	0.8367	0.8427	0.847	0.8091	0.8469
StackExchange	post-upvote (AUC ↑)	0.8853	0.6883	0.8817	0.6603	0.8896	0.7045	0.8861	0.6798	0.8845
	user-churn (AUC ↑)	0.8645	0.8528	0.867	0.856	0.8664	0.8657	0.8558	0.8485	0.8796

Figure 8: Raw AUC values for 4DBInfer classification tasks used in producing Figure 3(*right*).

Timing considerations. For many baselines, direct timing comparisons are elusive because of different computing environments and other confounds. That being said, we have confirmed that a single training run involving 4DBInfer baseline models described above operates on the scale of $10^3 - 10^4$ seconds. Combined with the 100-fold hyperparameter sweep (e.g., covering number of layers, hidden dimension, learning rate, etc.) needed to achieve reported results, the overall budget enters the $10^5 - 10^6$ second range. Meanwhile on comparable machines (a single NVIDIA 4080 GPU with 32GB memory) total RDBLearn latency with the same benchmarks is on the order of 10^2 seconds. This latency can be improved with further optimizations, but doing so lies outside the scope of the present work.

F Technical Proofs

Proposition F.1. *If σ and* agg *are positively homogeneous functions, we initialize embeddings using (4), and $\{w_0^{(h)}, w^{(h)}\} \in \mathbb{R}_+ \ \forall h$, then without loss of generality (5) can be reparameterized with no internal weights.*

Proof: We begin by examining two special cases and assuming that σ and agg are positively homogeneous of degree 1. First, if $\mu_v^{(h)} = 0$, then the node v update from (5) is equivalent to

$$\mu_v^{(h+1)} = \sigma\left(\text{agg}\left[\left\{w^{(h)}\mu_u^{(h)}\right\}_{u\in\mathcal{N}_v^{(h)}}\right]\right) = w^{(h)}\sigma\left(\text{agg}\left[\left\{\mu_u^{(h)}\right\}_{u\in\mathcal{N}_v^{(h)}}\right]\right) \tag{10}$$

since both σ and agg are positively homogeneous and $w^{(h)} \geq 0$ by assumption. Alternatively, if instead $\mathcal{N}_v^{(h)} = \emptyset$, meaning node v has no neighbors at point h on the meta-path, then

$$\mu_v^{(h+1)} = \sigma\left(w_0^{(h)}\mu_v^{(h)}\right) = w_0^{(h)}\sigma\left(\mu_v^{(h)}\right). \tag{11}$$

By definition of any H-hop meta-path and our corresponding meta-path GNN, we execute (5) H times. Combined with the proposed initialization strategy given by (4) and assumption of no loops along meta-paths mentioned below (5), every propagation step that results reduces to either (10) or (11). To see this, note that at any step h along a meta-path, by definition $\mathcal{N}_v^{(h)}$ only depends on a single active edge-type for all v, so any given node can only have neighbors once along a meta-path; elsewhere the update reduces to (11). And for a node having neighbors at a specific step along a meta-path, the corresponding node embedding prior to seeing neighbors will be zero by virtue of the initialization scheme. In this way, when neighbors do occur, (10) prevails. Hence the final embeddings produced at each node associated with rows of $\boldsymbol{X}$ will be compositions of (10) and (11). Given that a composition of positively homogeneous functions is also positively homogeneous, all weights can be pulled out in front without loss of generality. ■

[7] https://data.stackexchange.com/

Proposition F.2. *Assume $\pi : \mathbb{R}^\kappa \to \mathbb{R}$ is affine, with weights in general position. Then for any $n > \kappa$, there exists a fixed ICL decoder f_θ such that*

$$\left|y_{test} - f_\theta\big(\{\boldsymbol{x}_{i:}, y_i\}_{i=1}^n, \boldsymbol{x}_{test}\big)\right| = 0 \tag{12}$$

with probability one over datasets $\mathcal{D}(\pi)$ generated according to Definition 4.1. Meanwhile, for any $r < d$, and any possible fixed encoder-decoder pair $\{g_\phi, f'_\theta\}$, we have

$$\left|y_{test} - f'_\theta\big(\{\boldsymbol{z}_{i:}, y_i\}_{i=1}^n, \boldsymbol{z}_{test}\big)\right| > 0, \quad \boldsymbol{z} = g_\phi(\boldsymbol{x}) \in \mathbb{R}^r \tag{13}$$

with probability at least $1 - \binom{r}{k}/\binom{d}{k}$ over the same data generative distribution.

Proof: We split the proof into two parts.

Establishing (12). By assumption, the stacked ICL samples are generated as $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + b = \widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{w}} + b$, where $\widetilde{\boldsymbol{w}} \in \mathbb{R}^\kappa$ and $b \in \mathbb{R}$ are affine model parameters associated with π. Meanwhile $\boldsymbol{w}$ is simply $\widetilde{\boldsymbol{w}}$ padded with $d - \kappa$ zeros. From here, we first assume the $\kappa < n \leq d$. Let $\bar{\boldsymbol{X}}_n$ denote any set of n columns selected from $\boldsymbol{X}$. Per the sampling process used to generate $\boldsymbol{X}$, it follows that $\text{rank}[\bar{\boldsymbol{X}}_n] = n$ almost surely for any such $\bar{\boldsymbol{X}}_n$; for reference, this is equivalent to the condition $\text{spark}[\boldsymbol{X}] = n + 1$ almost surely (Donoho & Elad, 2003). Note that if it were that $\text{rank}[\bar{\boldsymbol{X}}_n] < n$ with non-negligible probability, then it must be that $p(\boldsymbol{X})$ has unbounded density on a subspace within $\mathbb{R}^d$, which is disallowed by construction.

Then, by extension of Lemma 2 from Wipf & Rao (2004), it follows that $\boldsymbol{y} - b = \boldsymbol{X}\boldsymbol{w}$ is the unique equality involving a sparse weight vector satisfying $\|\boldsymbol{w}\|_0 < n$. Hence the decoder can check each combination of n columns of $\boldsymbol{X}$ extracted from ICL samples to see if a sparse $\boldsymbol{w}$ vector allows for reconstructing observable labels $\boldsymbol{y}$ (note that there are more efficient ways to obtain $\boldsymbol{w}$ with additional assumptions; however, this is not necessary here for the proof). As any vector $\boldsymbol{w}$ so-obtained is unique and aligned with the ground-truth process, the model can then predict y_{test} label at any test point $\boldsymbol{x}_{\text{test}}$.

Lastly, if $n \geq d$, then $\text{rank}[\boldsymbol{X}] = d$. Hence the ground-truth generative weights satisfy $\boldsymbol{w} = \boldsymbol{X}^\dagger \boldsymbol{y}$ such that again, perfect estimation of new test points is possible as before.

Establishing (13). The distribution of $\boldsymbol{X}$ is the same for each generated dataset (only the distribution of $\boldsymbol{y}$ changes by design). With an r-dimensional output, q_θ can reconstruct at most r-dimensions of $\boldsymbol{X}$. Per the assumed generative process, the proportion of datasets whereby all κ columns fall within any group of r fixed columns of $\boldsymbol{X}$ is $\binom{r}{k}/\binom{d}{k}$. Hence the encoder can achieve zero estimation error recovering in $\binom{r}{k}/\binom{d}{k}$ proportion of cases by y_{test} by reconstructing any set of r columns of $\boldsymbol{X}$, leaving a failure ratio of $1 - \binom{r}{k}/\binom{d}{k}$. But can we do any better than this?

Note that it is not possible to perfectly reconstruct any one coordinate of a given $\boldsymbol{x}$ from the others, i.e., there does not exist a function $\psi : \mathbb{R}^{d-1} \to \mathbb{R}$ such that $x_j = \psi(\boldsymbol{x}_{\setminus j})$ almost surely, where $\boldsymbol{x}_{\setminus j}$ denotes the elements of $\boldsymbol{x}$ excluding the j-th coordinate. (If such a function existed, then the density $p(\boldsymbol{x})$ would be unbounded.) Consequently, to achieve zero estimation error at test points, the encoder must be capable of reconstructing all dimensions of $\boldsymbol{x}$ associated with nonzero elements in $\boldsymbol{w}$. And since the encoder is required to stay fixed for all datasets, at best it can reconstruct r columns, and so the success ratio from above cannot be improved upon. ■

Proposition F.3. *Let $\mathcal{F}$ denote the set of Lipschitz continuous functions on $[0,1]^\kappa$. Then there exists an ICL decoder f_θ such that (excluding smaller-order terms) we have*

$$\sup_{\pi \in \mathcal{F}} \mathbb{E}_{\mathcal{D}(\pi)\sim p}\Big[y_{test} - f_\theta\big(\{\boldsymbol{x}_{i:}, y_i\}_{i=1}^n, \boldsymbol{x}_{test}\big)\Big]^2 = O\left(n^{-2/\kappa}\right), \tag{14}$$

where the data distribution p is given by Definition 4.1 with $\mathcal{X} = [0,1]$. In contrast, with $\kappa = 1$ and $y = \widetilde{x}$, we have

$$\inf_{f_\theta} \sup_{g_\phi \in \mathcal{B}} \mathbb{E}_{\mathcal{D}(\pi)\sim p}\Big[y_{test} - f_\theta\big(\{\boldsymbol{z}_{i:}, y_i\}_{i=1}^n, \boldsymbol{z}_{test}\big)\Big]^2 = \Theta\left(n^{-2/d}\right), \tag{15}$$

where $\boldsymbol{z} = g_\phi(\boldsymbol{x})$ represents an encoder selected from the set of bi-Lipschitz bijections $\mathcal{B}$ mapping $[0,1]^d \to [0,1]^d$.

Proof: The proof is segmented into two parts as follows.

Establishing (14). This bound follows by piecing together well-known results from learning theory. We begin by splitting into equal halves a given set of training samples $\{\boldsymbol{x}_{i:}, y_i\}_{i=1}^n$ from some $\mathcal{D}(\pi)$ with S fixed. The high-level strategy is to define a set of nearest-neighbor estimators with the first half, and then select from among these estimators by enlisting the second half. Existing sample complexity results can then be applied to produce the final result.

To this end, for each subset of κ elements from d total input feature dimensions, we define the 1-nearest-neighbor estimator

$$f_{S'}(\boldsymbol{x}) = y_{i^*(\boldsymbol{x},S')}, \quad \text{with } i^*(\boldsymbol{x}, S') = \arg\min_{i<\lceil n/2 \rceil} \left\| \boldsymbol{x}_{S'} - (\boldsymbol{x}_{i:})_{S'} \right\| \tag{16}$$

and subscript S' denoting that only elements of any $\boldsymbol{x}$ within this subset are included. We then form a selection function $f_\theta(\boldsymbol{x}) = f_{S^*}(\boldsymbol{x})$ where

$$S^* = \arg\min_{S' \in \binom{[d]}{\kappa}} \left[\sum_{i=\lceil n/2 \rceil}^{n} \left(y_i - f_{S'}(\boldsymbol{x}_{i:}) \right)^2 \right]. \tag{17}$$

Regardless of how each constituent $f_{S'}$ was originally constructed, the subsequent selector based on (17) follows the oracle inequality

$$\begin{aligned}
&\mathbb{E}\left[\left(y_{\text{test}} - f_{S^*}(\boldsymbol{x}_{\text{test}}) \right)^2 \middle| S, \{\boldsymbol{x}_{i:}, y_i\}_{i<\lceil n/2 \rceil} \right] \\
&\qquad \leq \min_{S'} \mathbb{E}\left[\left(y_{\text{test}} - f_{S'}(\boldsymbol{x}_{\text{test}}) \right)^2 \middle| S, \{\boldsymbol{x}_{i:}, y_i\}_{i<\lceil n/2 \rceil} \right] + O\left(\sqrt{\frac{\log \binom{d}{\kappa}}{n}} \right) \\
&\qquad \leq \mathbb{E}\left[\left(y_{\text{test}} - f_{S}(\boldsymbol{x}_{\text{test}}) \right)^2 \middle| S, \{\boldsymbol{x}_{i:}, y_i\}_{i<\lceil n/2 \rceil} \right] + O\left(\sqrt{\frac{\log \binom{d}{\kappa}}{n}} \right),
\end{aligned} \tag{18}$$

where the expectation is over $\{\boldsymbol{x}_{\text{test}}, y_{\text{test}}\}$ and $\{\boldsymbol{x}_{i:}, y_i\}_{i=\lceil n/2 \rceil}^n$ following dataset generative process from Definition 4.1 but for now with S fixed. (i.e., the samples used in (16) are also treated as fixed here). See for example Section 5.4 of Hajek & Raginsky (2021) for that standard and Hoeffding and union bounding process used to establish the first inequality in (18); the second inequality trivially follows from removing the min operator.

We next need to bound the r.h.s. expectation within (18). Adopting i^* as shorthand for $i^*(\boldsymbol{x}_{\text{test}}, S)$, we have

$$\left| y_{\text{test}} - f_S(\boldsymbol{x}_{\text{test}}) \right| = \left| y_{\text{test}} - y_{i^*} \right| \leq L \left\| (\boldsymbol{x}_{\text{test}})_S - (\boldsymbol{x}_{i^*})_S \right\|, \tag{19}$$

which follows from the assumed Lipschitz continuity of π, recalling that $y = \pi(\boldsymbol{x}_S)$ by definition. We next square both sides and take an expectation over the samples $\{\boldsymbol{x}_{i:}, y_i\}_{i<\lceil n/2 \rceil}$ used in (16) as well as $\boldsymbol{x}_{\text{test}}$. These operations lead to

$$\mathbb{E}\left[\left(y_{\text{test}} - f_S(\boldsymbol{x}_{\text{test}}) \right)^2 \middle| S \right] \leq L^2 \mathbb{E}\left[\left\| (\boldsymbol{x}_{\text{test}})_S - (\boldsymbol{x}_{i^*})_S \right\|^2 \middle| S \right] \leq O\left(n^{-2/\kappa} \right), \tag{20}$$

where the right-most inequality stems from standard properties of nearest neighbors (e.g., see Theorem 2.1 in Biau & Devroye (2015)). Hence we can insert (20) into (18) after taking the expectation of both sides of the former w.r.t. $\{\boldsymbol{x}_{i:}, y_i\}_{i<\lceil n/2 \rceil}$. And lastly, because the resulting bound also holds for any given S, it also hold for all $S \in \binom{[d]}{\kappa}$. Hence we arrive at

$$\mathbb{E}\left[\left(y_{\text{test}} - f_{S^*}(\boldsymbol{x}_{\text{test}}) \right)^2 \right] \leq O\left(n^{-2/\kappa} \right) + O\left(\sqrt{\frac{\log \binom{d}{\kappa}}{n}} \right), \tag{21}$$

with expectation over the entire $\mathcal{D}(\pi)$ generative process. This expression is dominated by the first term for $\kappa > 4$ as n becomes large, completing the proof.

Establishing (15). Per the specified setup, we have $y = \widetilde{x} = [g_\phi^{-1}(\boldsymbol{z})]_i$ for some coordinate i, where $\boldsymbol{z} = g_\phi(\boldsymbol{x})$. For now we will assume that $i = 1$. Because g_ϕ is assumed to be a bi-Lipschitz, g_ϕ^{-1} is also Lipschitz over all of its coordinates, including the first. Therefore $h(\boldsymbol{z}) = [g_\phi^{-1}(\boldsymbol{z})]_1$ is a Lipschitz continuous function over domain $[0,1]^d$. Additionally, because $p(\boldsymbol{x}) \geq p_{\min}$, if follows that $p(\boldsymbol{z}) \geq p_{\min}/L^d$, where L is the upper Lipschitz constant associated with g. From here, it has already been established that the minimax squared estimation error for such an h, with input distribution bounded away from zero almost surely on a compact domain, is $\Theta\left(n^{-2/d}\right)$. Additionally, searching for an unknown i incurs a modest additional cost such that the overall rate is the same. ■