# 4DBInfer: A 4D Benchmarking Toolbox for Graph-Centric Predictive Modeling on RDBs

Minjie Wang[*]
Amazon
Shanghai, China
minjiw@amazon.com

Quan Gan[*]
Amazon
Shanghai, China
quagan@amazon.com

David Wipf
Amazon
Shanghai, China

Zhenkun Cai
Amazon
Shenzhen, China

Ning Li[†]
Shanghai Jiaotong
University
Shanghai, China

Jianheng Tang[†]
The Hong Kong
University of Science
and Technology
Hong Kong, China

Yanlin Zhang[†]
Fudan University
Shanghai, China

Zizhao Zhang[†]
Tsinghua University
Beijing, China

Zunyao Mao[†]
Southern University
of Science and
Technology
Shenzhen, China

Yakun Song[†]
Shanghai Jiaotong
University
Shanghai, China

Yanbo Wang[†]
Peking University
Beijing, China

Jiahang Li[†]
The Hong Kong
Polytechnic
University
Hong Kong, China

Han Zhang[†]
Shanghai Jiaotong
University
Shanghai, China

Guang Yang
Amazon
Shenzhen, China

Xiao Qin
Amazon
Santa Clara, CA, USA

Chuan Lei
Amazon
Santa Clara, CA, USA

Muhan Zhang
Peking University
Beijing, China

Weinan Zhang
Shanghai Jiaotong
University
Shanghai, China

Christos
Faloutsos
Carnegie Mellon
University
Pittsburgh, PA, USA

Zheng Zhang
Amazon
Shanghai, China

## ABSTRACT

Given a relational database (RDB), how can we predict missing column values in some target table of interest? Although RDBs store vast amounts of rich, informative data spread across interconnected tables, the progress of predictive machine learning models as applied to such tasks arguably falls well behind advances in other domains such as computer vision or natural language processing. This deficit stems, at least in part, from the lack of established/public RDB benchmarks as needed for training and evaluation purposes. As a result, related model development thus far often defaults to tabular approaches trained on ubiquitous single-table benchmarks, or on the relational side, graph-based alternatives such as GNNs applied to a completely different set of graph datasets devoid of tabular characteristics. To more precisely target RDBs lying at the nexus of these two complementary regimes, we explore a broad class of baseline models predicated on: (i) converting multi-table datasets into graphs using various strategies equipped with efficient subsampling, while preserving tabular characteristics; and (ii) trainable models with well-matched inductive biases that output predictions based on these input subgraphs. Then, to address the dearth of suitable public benchmarks and reduce siloed comparisons, we assemble a diverse collection of (i) large-scale RDB datasets and (ii) coincident predictive tasks. From a delivery standpoint, we operationalize the above *four dimensions* (4D) of exploration within a unified, scalable open-source toolbox called *4DBInfer*. We conclude by presenting evaluations using 4DBInfer, the results of which highlight the importance of considering each such dimension in the design of RDB predictive models, as well as the limitations of more naive approaches such as simply joining adjacent tables. Our source code is released at https://github.com/awslabs/multi-table-benchmark.

[*]Equal contribution
[†]Work done during internship at Amazon

# 1 INTRODUCTION

Relational databases (RDBs) can be viewed as storing a collection of interrelated data spread across multiple linked tables. Of vast and steadily growing importance, the market for RDB management systems alone is expected to exceed $133 billion USD by 2028 [56]. Even so, while the machine learning community has devoted considerable attention to predictive tasks involving *single* tables, or so-called tabular modeling tasks [21, 49, 58], thus far efforts to widen the scope to handle *multiple* tables and RDBs still lags behind, despite the seemingly enormous potential of doing so. With respect to the latter, in many real-world scenarios critical features needed for accurately modeling a given quantity of interest are not constrained to a single table [9, 14], nor can be easily flattened into a single table via reliable/obvious feature engineering [15].

This disconnect between commercial opportunity and academic research focus can, at least in large part, be traced back to one transparent culprit: Unlike widely-studied computer vision [16], natural language processing [67], tabular [28], and graph [35] domains, established benchmarks for evaluating predictive ML models of RDB data are much less prevalent. This reality is an unsurprising consequence of privacy concerns and the typical storage of RDBs on servers with heavily restrictive access and/or licensing protections. With few exceptions (that will be discussed in later sections), relevant model development is instead predicated on surrogate benchmarks that branch as follows.

Along the first branch, sophisticated models that explicitly account for relational information are often framed as graph learning problems, addressable by graph neural networks (GNNs) [6, 29, 32, 37, 42, 45, 57, 66] or their precursors [78, 80, 81], and evaluated specifically on graph benchmarks [35, 43, 51]. The limitation here though is that performance is conditional on a fixed, pre-specified graph and attendant node/edge features intrinsic to the benchmark, not an actual RDB or native multi-table format. Hence the inductive biases that might otherwise lead to optimal performance on the original data can be partially masked by whatever process was used to produce the provided graphs and features. As for the second branch, emphasis is placed on tabular model evaluations that preserve the original format of single table data, possibly with augmentations collected from auxiliary tables. But here feature engineering and table flattening are typically prioritized over exploiting rich network effects as with GNNs [9, 14, 47, 48]. Critically though, currently-available head-to-head comparisons involving diverse candidate approaches representative of *both* branches on un-filtered RDB/multi-table data are insufficient for drawing clear-cut conclusions regarding which might be preferable and under what particular circumstances.

To address the aforementioned limitations and help advance predictive modeling over RDB data, in Section 2 we first introduce a generic supervised learning formulation across both inductive and transductive settings covering dynamic RDBs as commonly-encountered in practice. A given predictive pipeline is then specified by (i) a sampling/distillation operator which extracts information most relevant to each target label, followed by (ii) a trainable prediction model. In Section 3 we present a specific design space for these two components. For the former, we adopt a graph-centric perspective whereby distillation is achieved (either implicitly or explicitly) via graphs and sampled subgraphs extracted from RDBs. Meanwhile, for the latter we incorporate trainable architectures that represent strong exemplars drawn from *both* tabular and graph ML domains. We emphasize here that until more extensive benchmarking has been conducted, it is advisable not to prematurely exclude candidates from either domain, or hybrid combinations thereof. In this regard, Section 4 introduces a new suite of RDB benchmarks along with discussion of the comprehensive desiderata which leads to them. These include multiple diversity/coverage considerations across both (i) datasets and (ii) predictive tasks, while also resolving limitations of existing alternatives. Our 4DBInfer toolbox for pairing a so-called *2D* design space of baseline models from Section 3 and the *2D* benchmark coverage from Section 4 within a neutral combined *4D* evaluation setting is introduced in Section 5. And finally, Section 6 culminates with representative experiments conducted using 4DBInfer.

In tracing these endeavors, our paper consolidates the following contributions:

- **2D Space of Baselines:** On the *modeling side* we describe a *2D* design space with considerable variation in (i) graph construction/sampling operators and (ii) trainable predictor designs. The latter covers popular choices drawn from GNN and tabular domains, representative of both early and late feature fusion strategies. This diversity safeguards against siloed comparisons between pipelines of only a single genre, e.g., tabular, GNNs.

- **2D Space of Benchmarks:** On the *data side*, we introduce a *2D* suite of RDB benchmarking (i) datasets and (ii) tasks that are devoid of potentially lossy or confounding pre-processing that might otherwise skew performance in favor of one model class or another. These benchmarks also vary across key dimensions of scale (e.g., up to 2B RDB rows), source domain, RDB schema, and temporal structure.

- **4DBInfer Toolbox:** We operationalize the above via a *unified and scalable open-source toolbox* called 4DBInfer that facilitates direct head-to-head empirical comparisons across each dimension of baseline model and benchmarking task (and is readily extensible to accommodate new additions of either). Figure 1 depicts the combined *4D* exploration space of 4DBInfer, along with comparisons relative to existing RDB, tabular, and graph benchmarking work.

- **Empirical Support:** Experiments using 4DBInfer highlight the relevance of each of the proposed four dimensions of exploration to the design of successful RDB predictive models, as well as the limitations of more naive approaches such as simply joining adjacent tables.

# 2 PREDICTIVE MODELING ON RDBS

## 2.1 Relational Database Preliminaries

An RDB $\mathcal{D}$ [25] can be viewed as a set of $K$ tables denoted as $\mathcal{D} := \{T^k\}_{k=1}^{K}$, where $T^k$ refers to the $k$-th constituent table defined by a particular entity type. Each row of a table then represents an instance of the corresponding entity type (e.g., a user), while the columns contain relevant features of each such instance (e.g., user profile information). Such features are typically heterogeneous and may include real values, integers, categorical variables, text
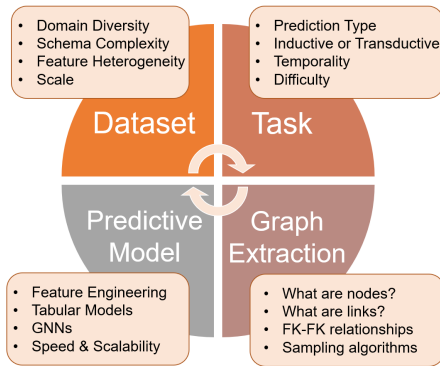
| "4D" Properties | OpenML [64] | OGB [34, 36] | HGB [51] | TGB [38] | RDBench [79] | CRLR [54] | RelBench [23] | 4DBInfer (Ours) |
|---|---|---|---|---|---|---|---|---|
| **1) Datasets** | | | | | | | | |
| Use raw data | ✓ | | | | ✓ | ✓ | ✓ | ✓ |
| Multiple Tables | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Heterogeneous Features | | | | | ✓ | ✓ | ✓ | ✓ |
| Billion-scale | | ✓ | | | | | | ✓ |
| **2) Tasks** | | | | | | | | |
| Transductive | | ✓ | ✓ | ✓ | | | | ✓ |
| Inductive | ✓ | | | | ✓ | ✓ | ✓ | ✓ |
| Temporal | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Entity Attr. Prediction | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Relationship Attr. Pred. | | | | | ✓ | ✓ | | ✓ |
| Key (or Link) Prediction | | ✓ | ✓ | ✓ | | | | ✓ |
| **3) Graphs** | | | | | | | | |
| Diverse extractions | | | | | | | | ✓ |
| **4) Predictive Models** | | | | | | | | |
| 4+ GNNs | | ✓ | ✓ | ✓ | ✓ | | | ✓ |
| 4+ Tabular ML | ✓ | | | | | Partial | | ✓ |

**Figure 1: 4DBInfer exploration dimensions.** Unlike prior benchmarking efforts (table columns on right), 4DBInfer considers an evaluation space with diversity across the 4D Cartesian product of (i) datasets, (ii) tasks, (iii) graph extractors, and (iv) predictive baselines. See Sections 3 and 4 (and in particular Section 4.3) for further details of table properties and assumptions.
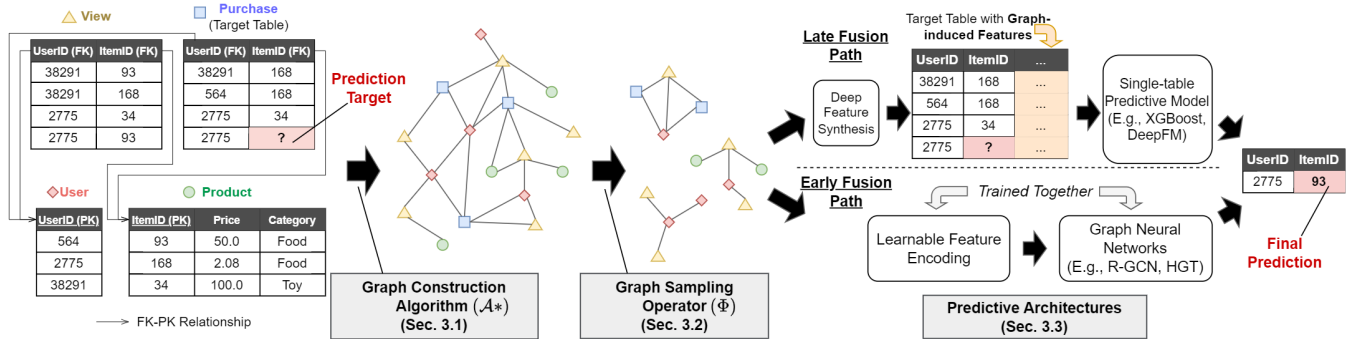


**Figure 2: 4DBInfer overview.** *Left*: First a (i) RDB dataset and (ii) task (i.e., predictive target here) are selected from among proposed benchmarks. *Middle*: Then a (iii) graph extractor/sampling operator is chosen which converts the RDB and task into subgraph chunks (*middle*). *Right*: Lastly a (iv) predictive model ingests these chunks, either through early or late feature fusion, to produce an estimate of the target values (*right*).

snippets, or time stamps among other things. We adopt $T_{i:}^k$ and $T_{:j}^k$ to reference the $i$-the row and $j$-th column of $T^k$ respectively. What establishes $\mathcal{D}$ as a *relational* database, as opposed to merely a collection of tables, is that certain table columns are designated as either *primary keys* (PKs) or *foreign keys* (FKs). A column $T_{:j}^k$ serves as a PK when each element is a unique index referencing a row of $T^k$, such as a user ID for example. In contrast, $T_{:j}^k$ is defined as a FK column if each $T_{ij}^k$ corresponds with a unique PK value referencing a row in *another* table $T^{k'}$ (generally $k' \neq k$, although this need not strictly be the case), with the only restriction being that all such indices within a given FK column must point to rows within the same table. In this way, the domain of any FK column is given by the corresponding PK column it references. Please see l.h.s. of Figure 2 for a simple RDB example.

## 2.2 Making Predictions over Dynamic RDBs

Generally speaking, RDBs are *dynamic*, with information regularly being added to or removed from $\mathcal{D}$. Hence if we are to precisely

define a predictive task involving an RDB, and particularly an inductive task, it is critical that we specify the RDB state during which a given prediction is to occur. For this reason, we refine our original RDB definition as $\mathcal{D}(s) := \{T^k(s)\}_{k=1}^K$, where $s \in \mathcal{S}$ defines the RDB *state* drawn from some set $\mathcal{S}$. Note that $\mathcal{S}$ could simply reflect counting indices (versions) such as the set of natural numbers; importantly though, each $s \in \mathcal{S}$ need not necessarily correspond with physical/real-world time per se, even if in some cases it may be convenient to assume so. This then leads to the following core objective:

> **Problem Statement:** *Using all relevant information available in $\mathcal{D}(s)$, predict an unknown RDB quantity of interest $T_{ij}^k(s)$ as uniquely specified by the tuple $\{s, k, i, j\}$, where $s$ determines the state, $k$ the table, and $\{i, j\}$ the table cell we wish to estimate.*

To illustrate, the unknown $T_{ij}^k(s)$ is represented by '?' on the l.h.s. of Figure 2. Ideally, we would like to closely approximate the

distribution $p\left(T_{ij}^k(s) \mid \mathcal{D}(s) \setminus T_{ij}^k(s)\right)$, meaning all other information in the RDB is fair game as conditioning variables governing our prediction at state $s$ of missing value $T_{ij}^k(s)$. Of course in practice it is neither feasible nor necessary to condition on the *entire* RDB given limited computational resources and the likely irrelevance of much of the stored data w.r.t. $T_{ij}^k(s)$. Hence our revised objective is to incorporate a sampling operator $\Phi$ defined such that

$$p\left(T_{ij}^k(s) \mid \Phi\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right]\right) \tag{1}$$
$$\approx p\left(T_{ij}^k(s) \mid \mathcal{D}(s) \setminus T_{ij}^k(s)\right),$$

where $\Phi\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right]$ represents a distillation of appreciable information in the RDB relevant to $T_{ij}^k(s)$. As a simple illustrative example, if

$$\Phi\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right] = T_{i:}^k(s) \setminus T_{ij}^k(s), \tag{2}$$

then all information in $\mathcal{D}(s)$ excluding the features in row $i$ of table $k$ are ignored when predicting $T_{ij}^k(s)$ and we recover a canonical tabular prediction task involving just a single table [21, 49, 58] More broadly though, $\Phi$ may be defined to select other rows of $T^k(s)$ (i.e., row $i' \neq i$ as used in recent cross-row tabular predictive models [20, 46, 60]), as well as information from other tables $T^{k'}(s)$ (with $k' \neq k$) that are linked to $T^k(s)$ through one or more FK relationships. Even other values in column $T_{:j}^k(s)$ can be incorporated when available, noting that a special case of this scenario can be used to rederive trainable variants of label propagation predictors [71].

## 2.3 High-Level Training and Inference Specs

We now describe training and inference in general terms under an inductive setup; the transductive case will trivially follow as a special case discussed below. We assume target table $k$ and target column $j$ are fixed to define a given predictive task. As such, each training instance is specified by only the tuple $\{s, i\}$, noting that target table row $i$ will often be a function of $s$ by design, e.g., as $s$ increments forward, additional rows with missing values for column $j$ may be added to $T^k(s)$. Let $\mathcal{S}_{tr}$ denote the set of states which have known training labels, and $\psi_{tr}(s)$ the corresponding set of specific indices with labels for each $s \in \mathcal{S}_{tr}$. Then for a given task defined by $k$ and $j$, along with a corresponding sampling operator $\Phi$, we seek to minimize the negative log-likelihood objective

$$\sum_{s \in \mathcal{S}_{tr}} \sum_{i \in \psi_{tr}(s)} -\log p\left(T_{ij}^k(s) \mid \Phi\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right]; \theta\right) \tag{3}$$

with respect to parameters $\theta$ that define the predictive distribution, e.g., a model of the conditional mean for regression problems, or logits for classification tasks, etc. The implicit assumption here is that, when conditioned on $\Phi\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right]$, each $T_{ij}^k(s)$ is roughly independent of one another for all $\{\{s, i\} : i \in \psi_{tr}(s), s \in \mathcal{S}_{tr}\}$; this implicit assumption forms the basis of empirical risk minimization [65].[1] However, it need *not* be the case that individual rows of $T^k(s)$ are independent of one another.

[1]There are alternatives to empirical risk minimization (ERM) for making predictions on RDBs, e.g., based on first-order logic [15, 27, 75]; however, for scalable, data-driven ML or deep learning solutions, ERM is a well-placed assumption.

Given some $\hat{\theta}$ obtained by minimizing (3), at test time we are presented with new tuples $\{\{s, i\} : i \in \psi_{te}(s), s \in \mathcal{S}_{te}\}$, from which we can compute $p\left(T_{ij}^k(s) \mid \Phi\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right]; \hat{\theta}\right)$ that ideally approximates the true distribution $p\left(T_{ij}^k(s) \mid \Phi\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right]\right)$. We remark that a transductive reduction of the above procedure naturally emerges when $s$ is fixed across both training and testing. More generally though, as $s$ increments $\mathcal{D}(s)$ may undergo significant changes, such as new rows appended to $T^k(s)$ (e.g., the 'Purchase' table in Figure 2), new labels/values added to the target column $T_{:j}^k(s)$, as well as arbitrary changes to other tables $T^{k'}(s)$ with $k' \neq k$.

## 3 DESIGN SPACE OF (GRAPH-CENTRIC) BASELINE MODELS

The general inductive learning framework from the previous section relies on two complementary components: (i) a sampling operator $\Phi$, and (ii) a parameterized predictive distribution as expressed in (3). Collectively, these amount to the first so-called 2D of our proposed 4DBInfer. For both scalability and conceptual reasons, we design the former to operate on graphs that can extracted from RDBs through multiple distinct strategies as summarized in Section 3.1. Subsequently, we will introduce the details of $\Phi$ itself in Section 3.2, followed by choices for predictive architectures in Section 3.3.

### 3.1 Converting RDBs to Graphs

A *heterogeneous graph* $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ [63] is defined by sets of node types $V$ and edge types $E$ such that $\mathcal{V} = \bigcup_{v \in V} \mathcal{V}^v$ and $\mathcal{E} = \bigcup_{e \in E} \mathcal{E}^e$, where $\mathcal{V}^v$ references a set of $|\mathcal{V}^v|$ nodes of type $v$, while $\mathcal{E}^e$ indicates a set of $|\mathcal{E}^e|$ edges of type $e$. Both nodes and edges can have associated features. Additionally, any heterogeneous graph can be generalized to depend on a state variable $s$ as $\mathcal{G}(s)$ analogous to $\mathcal{D}(s)$. The goal herein then becomes the establishment of some procedure or mapping $\mathcal{A}^*$ such that $\mathcal{G}(s) = \mathcal{A}^*[(\mathcal{D}(s)]$ for any given RDB of interest.

**Row2Node.** Perhaps the most natural and intuitive way to instantiate $\mathcal{A}^*$ is to simply treat each RDB row as a node, each table as a node type, and each FK-PK pair as a directed edge. Additionally, non-FK/PK column values are converted to node features assigned to the respective rows. Originally proposed in [15] with ongoing application by others [23, 77, 79], we refer to this approach as *Row2Node*; see Appendix G for further details.

**Row2N/E.** Importantly though, unlike prior work we do *not* limit 4DBInfer to a single selection for $\mathcal{A}^*$. The motivation for considering alternatives is straightforward: *Even if we believe that graphs are a sensible route for pre-processing RDB data, we should not prematurely commit to only one graph extraction procedure and the coincident downstream inductive biases that will inevitably be introduced.* To this end, as an alternative to Row2Node, we may relax the restriction that every row must be exclusively converted to a node. Instead, rows drawn from tables with more than one FK column can be selectively treated as typed edges, with the remaining non-FK columns designated as edge features. The intuition here is simply that tables with multiple FKs can be viewed as though they were natively a tabular representation of edges. We denote this

variant of $\mathcal{A}^*$ as *Row2N/E*, with full details and analysis deferred to Appendix G.

**Further Extensions.** And finally, we consider an extension of either Row2Node or Row2N/E designed to produce additional edges beyond those based on known FK-PK pairs. Motivated by practical use cases (as reflected in the benchmarks we will introduce later), the high-level idea is to introduce dummy tables with PK columns matched with select columns in original RDB tables. The latter are now treated as an additional set of *pseudo* FKs, which pair with dummy table PKs to form new typed edges/joins; notably, these may either be intra- or inter-table joins. Again, please see Appendix G for details of this approach and its advantages, which to the best of our knowledge, has not been addressed in prior work.

## 3.2 Graph-based Sampling Operator $\Phi$

In principle, the sampling operator $\Phi$ need not be explicitly predicated on an extracted graph. However, provided we do not restrict ourselves to *a particular fixed graph upfront*, we are not beholden to any one graph-specific inductive bias. In this way (with some abuse of notation) we instantiate $\Phi$ as

$$\Phi\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right] \equiv \Phi\left[\mathcal{A}^*\left[\mathcal{D}(s)\right] \setminus T_{ij}^k(s)\right]$$
$$= \Phi\left[\mathcal{G}(s) \setminus T_{ij}^k(s)\right], \quad (4)$$

where $\mathcal{A}^*$ is an RDB-to-graph mapping such as described in Section 3.1 (and Appendix G), $\mathcal{G}(s) = \mathcal{A}^*\left[\mathcal{D}(s)\right]$ represents the extracted graph, and the exclusion operator '\' now simply removes the node feature attribute associated with $T_{ij}^k(s)$ from $\mathcal{G}(s)$. We may now select from among the wide variety of scalable graph sampling methods for finalizing $\Phi$ [7, 10, 32, 73, 76, 83] while specifying the effective receptive field, meaning the number of hops (or tables) away from the target associated with $T_{ij}^k(s)$ from which information is collected. One reasonable choice is to match the receptive field to the RDB schema width, i.e., the maximal number of PK-FK hops needed to reach any other table from the target table. Whatever the choice though, the output of $\Phi$ will be a subgraph of $\mathcal{G}(s)$ containing the target node corresponding to row $T_{i:}^k(s)$.

Overall, provided we allow for diversity of graph extraction and sampling, then many classical multi-table data augmentation methods can be recast in this way. For example, joining a target table $k$ with all features from tables that can be reached by a single FK-PK join can be achieved using single-hop neighbor sampling applied to $\mathcal{G}(s)$. And for one-to-many joins (i.e., many FKs pointing to a single PK in the target table) it is a common feature engineering practice to just randomly choose a single element [9]; likewise, neighbor sampling over graphs can be optionally set to achieve the equivalent [32].

## 3.3 Trainable Predictive Architectures

At a high-level, once granted $\Phi$ we sub-divide candidate architectures for instantiating the predictive distribution from (3) based on what can be loosely referred to as early versus late feature fusion.

**Late Fusion.** In the context of RDB-specific modeling, we reserve *late fusion* to delineate models whereby parameter-free feature augmentation is adopted to produce a fixed-length, potentially high-dimensional feature vector associated with each target that is,

only then, used to train a high-capacity base model (with parameters $\theta$) such as those commonly applied to tabular data.[2] For the initial feature augmentation step, we lean on the Deep Feature Synthesis (DFS) framework from [41] and our own extensions thereof for the following reasons:

- DFS is a powerful automated method for generating new features for an RDB by recursively combining data from related tables through aggregation, transformation, etc.
- Although motivated differently, DFS can be re-derived and generalized as a form of subgraph sampling from Section 3.2, followed by concatenated aggregations, as applied to graphs extracted via Row2Node or Row2Node+;
- Special cases of DFS include commonly-used multi-table augmentation and flattening schemes [22], as when paired with sampling limited to 1-hop, or more general multi-hop strategies such as FastProp from the getML package [26];[3]
- DFS can be applied with constraints on $s$ to avoid label leakage;
- DFS is in principle capable of handling large-scale RDBs. Please see Appendix D.1 for additional details regarding DFS and our enhanced implementation.

Then for a given target $T_{ij}^k$, this so-called late fusion pipeline produces a fixed-length feature vector $\boldsymbol{u}_{ij}^k :=$

$$\text{DFS}\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right] \equiv \text{Agg}\left(\Phi\left[\mathcal{G}(s) \setminus T_{ij}^k(s)\right]\right), \quad (5)$$

where Agg is an aggregation operator; see Appendix D.1 for specific choices. And in conjunction with (4), we can subsequently apply any tabular model to estimate the parameters of

$$p\left(T_{ij}^k(s) \mid \Phi\left[\mathcal{D}(s) \setminus T_{ij}^k(s)\right]; \theta\right) \quad (6)$$
$$\equiv p\left(T_{ij}^k(s) \mid \boldsymbol{u}_{ij}^k; \theta\right)$$

by minimizing (3) over training data. For diversity of tabular base predictors, including both tree- and deep-learning-based, we adopt **MLP**, **DeepFM** [31], **FT-Transformer** [30], **XGBoost** [8], and **AutoGluon (AG)** [3, 21], the latter representing a top-performing AutoML tool that ensembles over multiple constituent models.

**Early Fusion.** We next adopt *early fusion* to reference message-passing GNN-like architectures that produce trainable low-dimensional node embeddings (at least relative to late fusion) beginning from the very first model layer. More concretely, for a heterogeneous graph $\mathcal{G}$ (e.g., as extracted from an RDB) these embeddings can be computed as $\boldsymbol{h}_{i,\ell}^v =$

$$f\left(\left\{\left\{\left(\boldsymbol{h}_{i',\ell-1}^{v'}, vv'\right) : i' \in \mathcal{N}_i^{vv'}\right\} : v' \in \mathcal{N}^v\right\}, \right.$$
$$\left. \boldsymbol{h}_{i,\ell-1}^v; \theta\right),$$

where $\boldsymbol{h}_{i,\ell}^v$ denotes the embedding of node $i$ of type $v$ at GNN layer $\ell$. In this expression, $\mathcal{N}^v$ indicates the set of node types that neighbor nodes of type $v$, and $\mathcal{N}_i^{vv'}$ is the set of nodes of type $v'$ that neighbor node $i$ of type $v$. Moreover, we assume that there is a unique edge type $e \equiv vv'$ associated with each pair of node types $(v, v')$, as will

---

[2]This strategy is also sometimes referred to as *propositionalization* [47, 75].
[3]The getML package also advertises other featurization algorithms for RDBs; however, these are not open-sourced nor published, and details are unavailable.

always be the case for the graphs extracted from RDBs that we focus on here (note also that the edge type $vv'$ is included within the inner-most set definition to differentiate each element within the outer-most set construction). Meanwhile, $f$ is a permutation-invariant function [74] over sets (with parameters $\theta$), acting to aggregate or fuse information from all neighbors of connected node types at each layer. At the output layer, the embeddings produced via (7) can be applied to making node-wise predictions, which translates into predictions of target values in column $T_{:j}^k$.

For implementing $f$ we adopt the popular heterogeneous ar-chitectures **R-GCN** [57], **R-GAT** [6], **HGT** [37], and **R-PNA** [13]. Note that we specifically select R-PNA because its core principal neighbor aggregation (extended to heterogeneous graphs) bears considerable similarities to DFS aggregators. In all cases the result-ing output layer embeddings will generally depend on which $\mathcal{A}^*$ is used for graph construction (see Appendix D.2 for further details).

## 3.4 Contextualization w.r.t Prior Work

Although not necessarily framed directly as such, recent work applying predictive ML or deep learning to RDBs can often be interpreted (implicitly or explicitly) as a particular graph extrac-tor ($\mathcal{A}^*$) along with graph-centric sampling ($\Phi$) followed by early [4, 33, 75, 77] or late fusion [9, 41, 47, 48] per the formulation out-lined herein.[4] However, there do not as of yet exist systematic comparisons among different pairings of available components (or different graph extraction approaches), nor in most cases is there available code for doing so. In particular, while late fusion-based models (per our terminology) mostly dominate ML solutions on RDBs thus far, more recent GNN-based alternatives (from the early fusion camp) are rarely actually pitted against the strongest incum-bents, and vice versa.

As a representative example, the recent RDB benchmarking work from [79] compares GNNs (with graphs from Row2Node) only against tabular baselines involving single tables and 1-hop table joins, not more advanced late fusion approaches like DFS. Con-versely, a strong late fusion approach from getML involving more sophisticated joins has recently been compared with GNNs [39], but only against one simple homogeneous GCN architecture [44] that is far from SOTA.

## 4 A NEW SUITE OF RDB BENCHMARKS

We now introduce RDB benchmarks that can be applied to eval-uating the efficacy of candidate predictive models such as those described in Section 3. However, first we discuss why existing benchmarks are not sufficient, followed by a more precise defini-tion of what actually constitutes a benchmark for our purposes. We conclude this section by describing our selection desiderata and specific benchmark choices that adhere to them.

## 4.1 Why New RDB Benchmarks

On the tabular side, there exist countless benchmarks covering ev-ery conceivable scenario; however, these are predominantly *single-table* datasets, e.g., widely-used Kaggle data [40]. In contrast, on

the relational side, benchmarks are often predicated on extracted graphs (often from limited domains such as citation networks) and pre-processed node features that may have already filtered away useful information [35, 43, 51]. As such, relative performance of candidate models is contingent on what information is available in these graphs and any sub-optimality therein, not actually the original data source. As a simple representative example, on the widely-studied Open Graph Benchmark (OGB) [35], many of the graph datasets were formed from curated citation networks with fixed text embeddings as node features. In this case, researchers have recently found that by reverting back to the original data sources and text features, vastly superior node classification accu-racy is possible [11]. Hence the original benchmarks were implicitly imposing an arbitrary constraint relative to the raw data itself, and the same can apply to imposed graph structure.

As for real-world datasets involving actual multi-table RDB data in its native form, available public benchmarks are somewhat lim-ited and narrow in scope. These include RDBench [79], RelBench [23], and the CTU Prague Relational Learning Repository (CRLR) [54]. However, as of the time of this submission, RelBench con-stitutes only two datasets, relies on Row2Node, and presents no experiments of any kind; see Appendix E for further differences between RelBench and our work. As for RDBench and CRLR, these are composed mostly of small datasets, e.g., with less than 1000 labeled instances, which is far surpassed by the size of typical real-world RDBs (see Section 4.3 below for further details). Additionally, among the recent model-driven works targeting predictive ML or deep learning on RDBs [4, 9, 24, 33, 50, 75, 77], there exists no con-sistent set of diverse data and tasks for empirical comparisons, and as alluded to previously, for most there is no available software allowing others to follow suit.

## 4.2 Definition of an RDB Benchmark

Before presenting our desiderata and dataset/task selections, it is helpful to first define what constitutes an RDB benchmark herein:

**Definition 4.1.** We define an *RDB benchmark*, denoted $\mathcal{B}$, as

$$\mathcal{B} := \{\, \{\mathcal{D}_{tr}, \mathcal{I}_{tr}\}, \{\mathcal{D}_{val}, \mathcal{I}_{val}\}, \{\mathcal{D}_{te}, \mathcal{I}_{te}\}, \ j, \ k \,\},$$

$$\text{where } \mathcal{D}_{spl} \quad := \quad \{\mathcal{D}(s)\}_{s \in \mathcal{S}_{spl}}, \tag{7}$$

$$\mathcal{I}_{spl} \quad := \quad \big\{\{i, s\} : i \in \psi_{spl}(s), s \in \mathcal{S}_{spl}\big\}$$

for all splits labeled $spl \in \{tr, \ val, \ te\}$ that reference training, validation, and testing respectively. For each such split, $\mathcal{D}_{spl}$ in-cludes the database contents at every state $s$ within the set $\mathcal{S}_{spl}$.[5] Meanwhile $\mathcal{I}_{spl}$ contains, for each state $s$ all of the indices $i$ of rows containing the target we wish to predict in column $j$ of table $k$, where $\psi_{spl}(s)$ specifies the set of such indices for each $s$.

By design, we may readily train baseline models via (3) using $\{\mathcal{D}_{tr}, \mathcal{I}_{tr}\}$ and task specification $\{j, k\}$, while using $\{\mathcal{D}_{val}, \mathcal{I}_{val}\}$ for hyperparameter tuning and model development, reserving $\{\mathcal{D}_{te}, \mathcal{I}_{te}\}$ for final performance evaluations. We also note that Definition 4.1 accommodates both inductive and transductive learning tasks depending on how $\mathcal{D}(s)$, the sets $\mathcal{S}_{spl}$, and point-to-set mappings

---

[4]There also exist feature augmentation methods based on reinforcement learning that fall outside of our current scope [24, 50]; moreover, scalability and sample-efficiency could pose challenges for such cases.

[5]There may be considerable redundancy across $s$ that can naturally be exploited for efficient storage; however, at least conceptionally the notion here is to have access to all relevant $\mathcal{D}(s)$ for each split.

$\psi_{spl}$ are defined. Either way, these items are each carefully specified to avoid label leakages, which otherwise represent a significant risk when facing the subtleties of real-world RDBs; see Appendix A.5 for a practical case study that exemplifies how label leakages can unexpectedly occur.

## 4.3 Benchmark Desiderata and Composition

To increase the chances that strong benchmark performance correlates with strong performance on future real-world application data, it is important to form each $\mathcal{B}$ so as to achieve adequate diversity or coverage across both (i) datasets and (ii) tasks. With this in mind, on the *dataset side* our selection criteria are as follows:

- **Availability:** Some otherwise promising public multi-table datasets currently disallow use for research publications [2, 12].
- **Large-scale:** Real-world RDBs can involve billions of rows.
- **Domain diversity:** We seek datasets from diverse domains spanning e-commerce, advertising, social networks, etc.
- **Schema diversity:** Schema width, # tables, # of rows, etc.
- **Temporal**: Realistic RDBs tend to vary over time.

Meanwhile, on the *task side* we have:

- **Loss type:** Regression, classification, or ranking;
- **Learning type:** Inductive versus transductive;
- **Proximity to real-world :** Tasks are chosen to reflect practical business scenarios.
- **Meaningful difficulty**: Poorly chosen tasks where informative features are lacking can lead to meaningless comparisons. Conversely, tasks involving auxiliary features that are simple functions of target labels may be trivially easy. In real-world scenarios, avoiding these extremes may be non-obvious; see Appendix E for representative case studies.

Based on these desiderata covering our proposed 2D dataset and task space underpinning 4DBInfer, we have curated a representative set of RDB benchmarks adhering to Definition 4.1. These are summarized in Table 1 along with distinguishing characteristic properties, with further details deferred to Appendix A. The datasets include: **AVS** [18], Outbrain (**OB**) [53], Diginetica (**DN**) [17], RetailRocket (**RR**) [84], Amazon Book Reviews (**AB**) [55], StackExchange (**SE**) [61], **MAG** [59], and **Seznam** (**SZ**) [54]. As for selected tasks on top of these datasets, please again see Table 1 and Appendix A.

Additionally, general comparisons with existing benchmarks are presented in Figure 1, where 4DBInfer displays a distinct advantage in terms of the four overall dimensions we have proposed warrant coverage. As shown in the figure (r.h.s.), relevant existing benchmarks include single-table tabular (**OpenML** [64]), graph (**OGB** [34, 36], **HGB** [51], **TGB** [38]), and RDB (**RDBench** [79], **CRLR** [54], **RelBench** [23]). Note that entity attribute and key prediction correspond with node classification and link prediction in the graph ML literature, respectively. Additionally, we only consider node classification and link prediction for OGB (including OGB-LSC [34]), i.e., graph classification is less related to RDB predictive tasks, and exclude synthetic data from CRLR. For further reference, Appendix E contains a much broader set of candidate benchmarks that were excluded from 4DBInfer because of failure to adhere with one or more of the above selection criteria.

| Dataset | # Rows | Task | # Instances | Temporal |
|---------|--------|------|-------------|----------|
| **AVS** | 350M | Retention | 160K | ✓ |
| **OB** | 2B | CTR | 87K | ✓ |
| **DN** | 3.7M | CTR | 120K | ✓ |
|         |        | Purchase | 177K | ✓ |
| **RR** | 23M | CVR | 100K | ✓ |
| **AB** | 16M | Churn | 1.3M | ✓ |
|        |       | Rating | 100K | ✓ |
|        |       | Purchase | 1.1M | ✓ |
| **SE** | 6.1M | Churn | 337K | ✓ |
|        |       | Popularity | 386K | ✓ |
| **MAG** | 23M | Venue | 736K | |
|         |      | Citation | 1.3M | |
| **SZ** | 2.7M | Charge | 554K | ✓ |
|        |       | Prepay | 1.4M | ✓ |

**Table 1: Properties of 4DBInfer datasets. CTR = Click-through-rate prediction, CVR = Conversion rate prediction. Comprehensive task details are listed in Appendix A.**

## 5 BENCHMARK & BASELINE DELIVERY

To facilitate reproducible empirical comparisons using our proposed benchmarks from Section 4 across the baselines from Section 3 (as well as future/improved predictive models informed by initial results), we instantiate 4DBInfer as a unified, scalable open-sourced Python package.[6] This package offers a *no-code* user experience to minimize the effort of experimenting with various baselines over built-in or customized RDB datasets. This is achieved via a composable and modularized design whereby each critical data processing and model training step can be launched independently or combined in arbitrary order. Moreover, adding a new RDB dataset simply requires users to describe its metadata and the location to download the tables while the pipeline will automate the rest.

As for the critical step of graph sampling, 4DBInfer implements $\Phi$ using the GraphBolt open-source APIs from the Deep Graph Library [68], which facilitates sampling over graphs with billions of nodes, which is roughly tantamount to RDBs with billions rows. Our 4DBInfer toolbox also provides an enhanced implementation of the DFS algorithm to facilitate the large-scale datasets in our benchmark suite. Specifically, the existing open-source implementation FeatureTools [22] can only leverage a single-thread for cross-table aggregation, which can take weeks on some of our datasets. We substitute its execution backend with an SQL-based engine, which translates the feature metadata into SQLs for execution. The resulting solution shortens the DFS computation time to only several hours. See Appendix F for quantitative metrics.

---

[6]https://github.com/awslabs/multi-table-benchmark

## 6 EXPERIMENTS

We now apply our 4DBInfer toolbox to explore performance across the proposed 4D evaluation space defined by benchmarks (datasets and tasks) and baselines (graph extractors/samplers and base predictors) applied to them. We conduct standard feature preprocessing for all experiments such as whitening numeric values, imputing missing entries, embedding text and date/time fields, etc. Importantly, to respect the dynamic evolution of RDB datasets, we employ temporal graph sampling, which ensures that only information about preceding events are collected for making predictions (i.e., only information available at RDB state $s$ during which the prediction is being made). All results are collected using the best early-stopping model w.r.t. the validation splits to avoid overfitting.

As for baseline models, we explore early feature fusion (DFS-based, with join-path set to reach the farthest RDB tables, i.e., the schema width) and late feature fusion (GNN-based) as discussed in Section 3.3. For the latter, we evaluate the impact of graphs extracted via either Row2Node (**R2N**) or Row2N/E (**R2N/E**). Where appropriate we also consider including dummy tables; see Appendix B.1. Furthermore, to better calibrate w.r.t. widely-used alternatives, we include comparisons with two additional baselines:

- **Single:** Ignore all other tables in an RDB except the target table of interest and apply classical tabular models.
- **Join:** Collect information from tables adjacent to the target table in the schema graph, and then apply tabular models to the resulting feature-augmented table (analogous to a simpliefed form of 1-hop DFS).

### 6.1 Main Results

Table 2 displays our main results spanning both the space of baseline models (rows) and benchmarks (columns). While there is considerable detail and nuance associated with these performance numbers, several key points are worth emphasizing as follows:

(a) **Complex vs simple comparisons.** More complex DFS-based and GNN-based models usually outperform both the single-table and simple join models, indicating that relevant predictive information exists across a wider RDB receptive field (i.e., beyond adjacent tables). These results also highlight the need to consider diverse, relatively large-scale datasets, as prior work [79] involving much smaller scales has shown that simple joins can outperform GNNs.

(b) **Early vs late feature fusion.** Early feature fusion as instantiated via GNNs is generally preferable to late fusion through DFS-based models. That being said, DFS nonetheless remains a strong competitor on multiple benchmarks, particularly AVS-Retention, DN-CTR, and SE-Popularity. Moreover, because of its lean design relative to GNNs, late fusion may be especially favorable in low resource environments even if the accuracy is not necessarily superior.

(c) **Graph extraction method matters.** Among the 12 cases where GNNs perform well, 4 (OB-CTR, AB-Purchase, MAG-Venue, MAG-Cite) have strong bias towards Row2N/E while 3 significantly favor Row2Node (DN-CTR, AB-Churn, SE-Popularity). Hence further exploration along the graph extraction dimension is warranted.

(d) **Task specific dependencies.** GNNs are significantly preferable for predicting foreign keys, which is analogous to link prediction tasks in the graph ML literature. The latter typically benefits from more complex structural signals such as common neighbors, for which GNNs are arguably more equipped to exploit.

**Summary.** In one way or another, all of the points above *highlight the value of considering all four dimensions of our proposed 4D exploration space*, namely, the potential consequences of variability across **dataset (a,b,c)**, **task (d)**, **graph extractor (c)**, and **base predictor (a,b,d)**. Even so, our preliminary comparisons so-obtained crown no unequivocal front-runner across all scenarios, showcasing the need for such benchmarking on realistic RDB tasks in the first place. And quite plausibly, high-performant solutions may actually lie at the boundary between tabular and graph ML worlds. Either way, reliably establishing such trends hinges on native RDB evaluations that do not (to the extent possible) a priori favor one approach over another, e.g., results conditional on only one specific pre-processed graph or feature engineering technique, etc.

### 6.2 Ablations

We conclude our empirical study by summarizing various ablations, with full descriptions deferred to Appendix B.

- **Stronger GNN model.** We examine the extent to which more recent GNN architectures might further boost performance. For this purpose, we conduct experiments using neural common neighbors (NCN) [69], a powerful architecture specifically targeting link prediction. As detailed in Appendix B.2, on 7 of 8 benchmarks related to key or relationship attribute prediction, NCN improves upon all of the baselines in Table 2.
- **Use of dummy tables.** In Section 3.1 we described how the strategic use of dummy tables can lead to extracted graphs with additional inter- or intra-table edges. Appendix B.1 compares across identical settings with and without the use of such dummy tables; in many cases there is a significant performance impact, e.g., for R-GCN models using Row2Node on AVS-Retent the AUC drops from 0.5653 to 0.4761 without dummy tables (a similar drop also occurs when using Row2N/E).
- **Label propagation.** Finally, as alluded to in Section 2.2, it is possible to handle trainable generalizations of label propagation using the conceptual framework that underpins 4DBInfer. We explore this possibility in Appendix B.3, demonstrating that the judicious use of observable labels can positively influence performance by significant margins, e.g., without such use of labels the AUC can drop by over 0.10 on the OB-CTR task.

## 7 CONCLUSION

In this work we have introduced 4DBInfer, a flexible open-source tool designed to explore predictive modeling on RDBs along four influential dimensions, namely, (1) dataset, (2) task, (3) graph extraction, and (4) base predictive model. Moreover, we have empirically verified the relevance of these dimensions and the value of comparing strong models from both tabular and graph ML camps in a neutral setting.

| Dataset | | AVS | OB | DN | | RR | AB | | | SE | | MAG | | SZ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | | Retent. | CTR | CTR | Purch. | CVR | Churn | Rating | Purch. | Churn | Popul. | Venue | Cite | Charge | Prepay |
| Prediction Type | | RA | RA | RA | FK | RA | EA | RA | FK | EA | EA | EA | FK | EA | EA |
| Evaluation metric | | AUC↑ | AUC↑ | AUC↑ | MRR↑ | AUC↑ | AUC↑ | RMSE↓ | MRR↑ | AUC↑ | AUC↑ | Acc.↑ | MRR↑ | Acc.↑ | Acc.↑ |
| Induct. or Trans. | | Ind. | Ind. | Ind. | Ind. | Ind. | Ind. | Ind. | Ind. | Ind. | Ind. | Trans. | Trans. | Ind. | Ind. |
| Single | MLP | 0.5300 | N/A | N/A | N/A | N/A | 0.5000 | N/A | N/A | 0.5000 | 0.5079 | 0.2686 | N/A | 0.4375 | 0.5314 |
| | DeepFM | 0.5217 | N/A | N/A | N/A | N/A | 0.5000 | N/A | N/A | 0.4964 | 0.5078 | N/A | N/A | 0.4242 | 0.5294 |
| | FT-Trans | 0.5013 | N/A | N/A | N/A | N/A | 0.5000 | N/A | N/A | 0.4998 | 0.5124 | 0.2370 | N/A | 0.4367 | 0.5275 |
| | XGB | 0.5033 | N/A | N/A | N/A | N/A | 0.5000 | N/A | N/A | 0.5084 | 0.4968 | 0.2176 | N/A | 0.4483 | 0.5285 |
| | AG | 0.5350 | N/A | N/A | N/A | N/A | 0.5000 | N/A | N/A | 0.5000 | 0.5081 | 0.2547 | N/A | 0.4561 | 0.5145 |
| Join | MLP | 0.5618 | 0.4891 | 0.5450 | 0.0519 | 0.5097 | 0.5000 | 1.0570 | 0.0881 | 0.6024 | 0.8745 | 0.3267 | 0.4989 | 0.5692 | 0.6110 |
| | DeepFM | 0.5620 | 0.5109 | 0.5057 | 0.0502 | 0.4933 | 0.5000 | 1.0585 | 0.0873 | 0.5984 | 0.8764 | 0.2819 | 0.4506 | 0.5416 | 0.5915 |
| | FT-Trans | 0.5569 | 0.5203 | 0.5584 | 0.0612 | 0.4917 | 0.5000 | 1.0574 | 0.0919 | 0.6319 | 0.8670 | 0.2243 | 0.4918 | 0.5825 | 0.6319 |
| | XGB | 0.5271 | 0.5000 | 0.5340 | 0.0316 | 0.5000 | 0.5000 | 1.0550 | 0.0909 | 0.5820 | 0.8669 | 0.2195 | 0.0329 | 0.5878 | 0.6266 |
| | AG | 0.5432 | 0.4969 | 0.5207 | 0.0538 | 0.5096 | 0.5000 | 1.0501 | 0.0853 | 0.5820 | 0.8669 | 0.2571 | 0.0329 | 0.5938 | 0.6354 |
| DFS | MLP | 0.5690 | 0.5456 | 0.6944 | 0.0743 | 0.8181 | 0.6815 | 0.9847 | 0.1112 | 0.8326 | 0.8783 | 0.2887 | 0.4903 | 0.7554 | 0.8248 |
| | DeepFM | 0.5669 | 0.5289 | 0.7341 | 0.0635 | 0.8182 | 0.6667 | 0.9946 | 0.0845 | 0.8212 | 0.8821 | 0.2476 | 0.5760 | 0.7016 | 0.8092 |
| | FT-Trans | 0.5665 | 0.5360 | 0.7412 | 0.0582 | 0.8034 | 0.6765 | 0.9888 | 0.1191 | 0.8376 | 0.8749 | 0.3010 | 0.3635 | 0.7473 | 0.8162 |
| | XGB | 0.5469 | 0.5421 | 0.7219 | 0.0376 | 0.7906 | 0.6922 | 0.9972 | 0.0909 | 0.8251 | 0.8675 | 0.2202 | 0.0329 | 0.7600 | 0.8453 |
| | AG | 0.5665 | 0.5494 | 0.7219 | 0.0749 | 0.8008 | 0.7291 | 0.9829 | 0.0888 | 0.8396 | 0.8849 | 0.3208 | 0.0329 | 0.7731 | 0.8485 |
| R2N | R-GCN | 0.5578 | 0.6239 | 0.7273 | 0.3557 | 0.8470 | 0.7358 | 0.9639 | 0.1790 | 0.8558 | 0.8861 | 0.4336 | 0.7020 | 0.7917 | 0.8768 |
| | R-GAT | 0.5637 | 0.6146 | 0.6741 | 0.3595 | 0.8284 | 0.7410 | 0.9563 | 0.1546 | 0.8645 | 0.8853 | 0.4408 | 0.7072 | 0.8053 | 0.8954 |
| | R-PNA | 0.5606 | 0.6249 | 0.7011 | 0.3638 | 0.8366 | 0.7645 | 0.9615 | 0.1791 | 0.8664 | 0.8896 | 0.5119 | 0.6534 | 0.8000 | 0.8924 |
| | HGT | 0.5703 | 0.6260 | 0.6733 | 0.2207 | 0.8495 | 0.7551 | 0.9636 | 0.1325 | 0.8670 | 0.8817 | 0.4164 | 0.6768 | 0.7965 | 0.8805 |
| R2N/E | R-GCN | 0.5653 | 0.6271 | 0.7507 | 0.3691 | 0.8091 | 0.7207 | 0.9696 | 0.2503 | 0.8485 | 0.6798 | 0.4936 | 0.8065 | 0.7842 | 0.8731 |
| | R-GAT | 0.5638 | 0.6308 | 0.7320 | 0.3746 | 0.7536 | 0.7258 | 0.9657 | 0.3055 | 0.8528 | 0.6883 | 0.5119 | 0.794 | 0.8065 | 0.8963 |
| | R-PNA | 0.5608 | 0.6322 | 0.6414 | 0.3758 | 0.8427 | 0.7348 | 0.9675 | 0.252 | 0.8657 | 0.7045 | 0.5159 | 0.7716 | 0.7988 | 0.8847 |
| | HGT | 0.5630 | 0.6323 | 0.6672 | 0.2072 | 0.8342 | 0.7208 | 0.9663 | 0.2916 | 0.8560 | 0.6603 | 0.4692 | 0.7896 | 0.8071 | 0.8965 |

**Table 2: 4DBInfer is informative.** Performance results of baselines; the top-5 performing models on each dataset are shaded green - the darker, the better. For abbreviations, EA = Entity Attribute Prediction, RA = Relationship Attribute Prediction, FK = Foreign Key Prediction, Ind. = Inductive, Trans. = Transductive. And some entries are marked as 'N/A' because there are no features in the target table such that single table models cannot be applied.

## REFERENCES

[1] Sameer Agarwal, Kristin Branson, and Serge Belongie. 2006. Higher order learning with graphs. In *Proceedings of the 23rd international conference on Machine learning*. 17–24.

[2] KirillOdintsov Martin Kotek Anna Montoya, inversion. 2018. Home Credit Default Risk. https://kaggle.com/competitions/home-credit-default-risk

[3] AutoGluon 2023. https://auto.gluon.ai/scoredebugweight/tutorials/tabular_prediction/index.html

[4] Jinze Bai, Jialin Wang, Zhao Li, Donghui Ding, Ji Zhang, and Jun Gao. 2021. ATJ-Net: Auto-Table-Join Network for Automatic Learning on Relational Databases. In *Proceedings of the Web Conference 2021*. 1540–1551.

[5] Alain Bretto. 2013. Hypergraph theory. *An introduction. Mathematical Engineering. Cham: Springer* 1 (2013).

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

[7] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

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

[9] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *Proc. VLDB Endow.* 13, 9 (may 2020), 1373–1387. https://doi.org/10.14778/3397230.3397235

[10] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 257–266.

[11] Eli Chien, Wei-Cheng Chang, Cho-Jui Hsieh, Hsiang-Fu Yu, Jiong Zhang, Olgica Milenkovic, and Inderjit S Dhillon. 2021. Node feature extraction by self-supervised multi-scale neighborhood prediction. *arXiv preprint arXiv:2111.00064* (2021).

[12] Will Cukierski Claudia Perlich, Vladimir Dubovskiy. 2014. KDD Cup 2014 - Predicting Excitement at DonorsChoose.org. https://kaggle.com/competitions/kdd-cup-2014-predicting-excitement-at-donors-choose

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

[14] Alexis Cvetkov-Iliev, Alexandre Allauzen, and Gaël Varoquaux. 2023. Relational data embeddings for feature enrichment with background information. *Machine Learning* 112, 2 (2023), 687–720.

[15] Milan Cvitkovic. 2020. Supervised learning on relational databases with graph neural networks. *arXiv preprint arXiv:2002.02046* (2020).

[16] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. 248–255.

[17] Diginetica 2016. https://competitions.codalab.org/competitions/11161

[18] DMDave, Todd B, and Will Cukierski. 2014. Acquire Valued Shoppers Challenge. https://kaggle.com/competitions/acquire-valued-shoppers-challenge

[19] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 456–467.

[20] Kounianhua Du, Weinan Zhang, Ruiwen Zhou, Yangkun Wang, Xilong Zhao, Jiarui Jin, Quan Gan, Zheng Zhang, and David P Wipf. 2022. Learning Enhanced Representation for Tabular Data via Neighborhood Propagation. *Advances in Neural Information Processing Systems* 35 (2022), 16373–16384.

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

[22] Featuretools 2023. https://www.featuretools.com/

[23] Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. 2023. Relational Deep Learning: Graph Representation Learning on Relational Databases. *arXiv preprint arXiv:2312.04615* (2023).

[24] Sainyam Galhotra, Yue Gong, and Raul Castro Fernandez. 2023. METAM: Goal-Oriented Data Discovery. *arXiv preprint arXiv:2304.09068* (2023).

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

[26] getML 2023. https://www.getml.com/

[27] Lise Getoor and Ben Taskar. 2007. *Introduction to statistical relational learning.* MIT press.

[28] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. 2019. An open source AutoML benchmark. *arXiv preprint arXiv:1907.00909* (2019).

[29] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning.* PMLR, 1263–1272.

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

[31] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).

[32] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* 30 (2017).

[33] Benjamin Hilprecht, Kristian Kersting, and Carsten Binnig. 2023. SPARE: A Single-Pass Neural Model for Relational Databases. *arXiv preprint arXiv:2310.13581* (2023).

[34] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. 2021. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430* (2021).

[35] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems* 33 (2020), 22118–22133.

[36] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. *arXiv preprint arXiv:2005.00687* (2020).

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

[38] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. 2023. Temporal graph benchmark for machine learning on temporal graphs. *arXiv preprint arXiv:2307.01026* (2023).

[39] Jankmeyer 2023. https://www.kaggle.com/code/jankmeyer/cora-relational-learning-vs-graph-neural-networks/notebook

[40] Kaggle 2024. https://www.kaggle.com/datasets

[41] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA).* IEEE, 1–10.

[42] Steven M. Kearnes, Kevin McCloskey, Marc Berndl, Vijay S. Pande, and Patrick Riley. 2016. Molecular graph convolutions: moving beyond fingerprints. *J. Comput. Aided Mol. Des.* 30, 8 (2016), 595–608.

[43] Arpandeep Khatua, Vikram Sharma Mailthody, Bhagyashree Taleka, Tengfei Ma, Xiang Song, and Wen-mei Hwu. 2023. IGB: Addressing The Gaps In Labeling, Features, Heterogeneity, and Size of Public Graph Datasets for Deep Learning Research. *arXiv preprint arXiv:2302.13522* (2023).

[44] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[45] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations* (Palais des Congrès Neptune, Toulon, France) *(ICLR '17).* Palais des Congrès Neptune, Toulon, France. https://openreview.net/forum?id=SJU4ayYgl

[46] Jannik Kossen, Neil Band, Clare Lyle, Aidan N Gomez, Thomas Rainforth, and Yarin Gal. 2021. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *Advances in Neural Information Processing Systems* 34 (2021), 28742–28756.

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

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

[49] Erin LeDell and Sebastien Poirier. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020. ICML.

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

[51] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. 2021. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining.* 1150–1160.

[52] MariaDB 2015. https://mariadb.com/kb/en/database-normalization-5th-normal-form-and-beyond/

[53] mjkistler, Ran Locar, Ronny Lempel, RoySassonOB, Rwagner, and Will Cukierski. 2016. Outbrain Click Prediction. https://kaggle.com/competitions/outbrain-click-prediction

[54] Jan Motl and Oliver Schulte. 2015. The CTU prague relational learning repository. *arXiv preprint arXiv:1511.03086* (2015).

[55] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 188–197. https://doi.org/10.18653/v1/D19-1018

[56] Research Industry Network 2023. https://linkedin.com/pulse/2031-relational-database-management/

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

[58] Ravid Shwartz-Ziv and Amitai Armon. 2022. Tabular data: Deep learning is not all you need. *Information Fusion* 81 (2022), 84–90.

[59] Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June (Paul) Hsu, and Kuansan Wang. 2015. An Overview of Microsoft Academic Service (MAS) and Applications. In *Proceedings of the 24th International Conference on World Wide Web* (Florence, Italy) *(WWW '15 Companion).* Association for Computing Machinery, New York, NY, USA, 243–246. https://doi.org/10.1145/2740908.2742839

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

[61] StackExchange Data Explorer [n. d.]. https://data.stackexchange.com/

[62] Xiangguo Sun, Hongzhi Yin, Bo Liu, Hongxu Chen, Jiuxin Cao, Yingxia Shao, and Nguyen Quoc Viet Hung. 2021. Heterogeneous hypergraph embedding for graph classification. In *Proceedings of the 14th ACM international conference on web search and data mining.* 725–733.

[63] Yizhou Sun and Jiawei Han. 2013. Mining heterogeneous information networks: a structural analysis approach. *ACM SIGKDD Explorations Newsletter* 14, 2 (2013), 20–28.

[64] Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. 2014. OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter* 15, 2 (2014), 49–60.

[65] Vladimir Vapnik. 1991. Principles of risk minimization for learning theory. *Advances in Neural Information Processing systems* 4 (1991).

[66] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings.* OpenReview.net, Vancouver, BC, Canada. https://openreview.net/forum?id=rJXMpikCZ

[67] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).

[68] Minjie Wang, Da Zheng, Zihao Ye, Quan Gan, Mufei Li, Xiang Song, Jinjing Zhou, Chao Ma, Lingfan Yu, Yu Gai, Tianjun Xiao, Tong He, George Karypis, Jinyang Li, and Zheng Zhang. 2019. Deep Graph Library: A Graph-Centric, Highly-Performant Package for Graph Neural Networks. *arXiv preprint arXiv:1909.01315* (2019).

[69] Xiyuan Wang, Haotong Yang, and Muhan Zhang. 2023. Neural Common Neighbor with Completion for Link Prediction. *arXiv preprint arXiv:2302.00890* (2023).

[70] Yuxin Wang, Quan Gan, Xipeng Qiu, Xuanjing Huang, and David Wipf. 2023. From hypergraph energy functions to hypergraph neural networks. In *International Conference on Machine Learning.*

[71] Yangkun Wang, Jiarui Jin, Weinan Zhang, Yongyi Yang, Jiuhai Chen, Quan Gan, Yong Yu, Zheng Zhang, Zengfeng Huang, and David Wipf. 2022. Why propagate alone? Parallel use of labels and features on graphs. *International Conference on Learning Representations* (2022).

[72] Chaoqi Yang, Ruijie Wang, Shuochao Yao, and Tarek F. Abdelzaher. 2020. Hypergraph Learning with Line Expansion. *ArXiv* abs/2005.04843 (2020).

[73] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining.* 974–983.

[74] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. *Advances in neural information processing systems* 30 (2017).

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

[76] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the Depth and Scope of Graph Neural Networks. *Advances in Neural Information Processing Systems* 34 (2021).

[77] Han Zhang, Quan Gan, David Wipf, and Weinan Zhang. 2023. Gfs: Graph-based feature synthesis for prediction over relational databases. *arXiv preprint arXiv:2312.02037* (2023).

[78] Xinhua Zhang and Wee Lee. 2006. Hyperparameter learning for graph based semi-supervised learning algorithms. *Advances in neural information processing systems* 19 (2006).

[79] Zizhao Zhang, Yi Yang, Lutong Zou, He Wen, Tao Feng, and Jiaxuan You. 2023. RDBench: ML Benchmark for Relational Databases. *arXiv preprint arXiv:2310.16837* (2023).

[80] Dengyong Zhou, Olivier Bousquet, Thomas Lal, Jason Weston, and Bernhard Schölkopf. 2003. Learning with local and global consistency. *Advances in neural information processing systems* 16 (2003).

[81] Xiaojin Jerry Zhu. 2005. Semi-supervised learning literature survey. *University of Wisconsin-Madison Department of Computer Sciences* (2005).

[82] J.Y. Zien, M.D.F. Schlag, and P.K. Chan. 1999. Multilevel spectral hypergraph partitioning with arbitrary vertex sizes. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18, 9 (1999), 1389–1399. https://doi.org/10.1109/43.784130

[83] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in Neural Information Processing Systems* 32 (2019).

[84] Roman Zykov, Noskov Artem, and Anokhin Alexander. 2022. Retailrocket recommender system dataset. https://doi.org/10.34740/KAGGLE/DSV/4471234

# A   DATASET AND TASK DESCRIPTIONS

In this section we provide comprehensive details pertaining to all datasets and tasks originally listed in Table 2. Additionally, for aggregated summary statistics/attributes across each dataset and task, please see Tables 3 and 4, respectively. If not specified, we split training, validation and testing samples according to their timestamps, to simulate real-world scenarios where the trained models will be evaluated over new observed samples. Another design choice is the prediction timestamp, which determines what information in the RDB is available at prediction time. By default, we use the timestamp of the target table as the prediction time, assuming that prediction needs to be made upon a new entry added to the target table. All the datasets have been released as part of the Python package dbinfer-bench:

```
pip install dbinfer-bench
```

It can then be loaded from Python:

```
import dbinfer_bench as dbb
```

| Dataset | # Tables | # Columns | # Rows |
|---------|----------|-----------|--------|
| AVS | 3 | 24 | 349,967,371 |
| Outbrain (OB) | 8 | 31 | 2,170,441,217 |
| Diginetica (DN) | 5 | 28 | 3,672,396 |
| RetailRocket (RR) | 3 | 11 | 23,033,676 |
| Amazon (AB) | 3 | 15 | 24,291,489 |
| StackExchange (SE) | 7 | 49 | 5,399,818 |
| MAG | 5 | 13 | 21,847,396 |
| Seznam (SZ) | 4 | 14 | 2,681,983 |

**Table 3: Statistics of each dataset.**

| Dataset | Task Description | Prediction Type | Metric | #Train / #Val / #Test |
|---------|------------------|-----------------|--------|----------------------|
| AVS | Customer Retention Prediction (Retent.) | Relationship Attribute | AUC↑ | 109,341 / 24,261 / 26,455 |
| Outbrain (OB) | Click-through-rate Prediction (CTR) | Relationship Attribute | AUC↑ | 69,709 / 8,715 / 8,718 |
| Diginetica (DN) | Click-through-rate Prediction (CTR) | Relationship Attribute | AUC↑ | 108,570 / 6,262 / 5,058 |
| | Purchase Prediction (Purch.) | Foreign Key | MRR↑ | 16,247 / 82,721 / 78,357 |
| RetailRocket (RR) | Conversion-rate Prediction (CVR) | Relationship Attribute | AUC↑ | 80,008 / 9,995 / 9,997 |
| Amazon (AB) | User Churn Prediction (Churn) | Entity Attribute | AUC↑ | 1,045,568 / 149,205 / 152,486 |
| | Rating Prediction (Rating) | Relationship Attribute | RMSE↓ | 78,485 / 7,762 / 13,492 |
| | Purchase Prediction (Purch.) | Foreign Key | MRR↑ | 78,485 / 387,914 / 677,211 |
| StackExchange (SE) | User Churn Prediction (Churn) | Entity Attribute | AUC↑ | 142,877 / 88,164 / 105,612 |
| | Post Popularity Prediction (Popul.) | Entity Attribute | AUC↑ | 308,698 / 38,587 / 38,588 |
| MAG | Venue Prediction (Venue) | Entity Attribute | Acc.↑ | 629,571 / 64,879 / 41,939 |
| | Citation Prediction (Cite) | Foreign Key | MRR↑ | 108,000 / 591,942 / 592,176 |
| Seznam (SZ) | Charge Type Prediction (Charge) | Entity Attribute | Acc.↑ | 443,276 / 55,410 / 55,410 |
| | Prepay Type Prediction (Prepay) | Entity Attribute | Acc.↑ | 1,151,620 / 143,952 / 143,953 |

**Table 4: Benchmark dataset and task details. Note that for foreign key prediction, the validation and test instances include the generated negative samples as well.**

## A.1   Acquire Valued Shoppers Challenge (AVS)

```
dataset = dbb.load_rdb_data('avs')
```

The Acquire Valued Shoppers Challenge [18] is a Kaggle dataset from an e-commerce platform. The dataset has three tables: a History table containing the history of each promotion offer given to a customer, an Offers table containing the information of the promotion offers themselves, and a Transactions table containing the transaction history between customers and products. The schema diagram is shown in Figure 3, where the timestamp column, primary keys, and foreign keys are indicated. Note that the Customers, Chain, Category, Company and Brand tables are dummy tables that do not exist natively, but are induced from the corresponding foreign key columns [15].
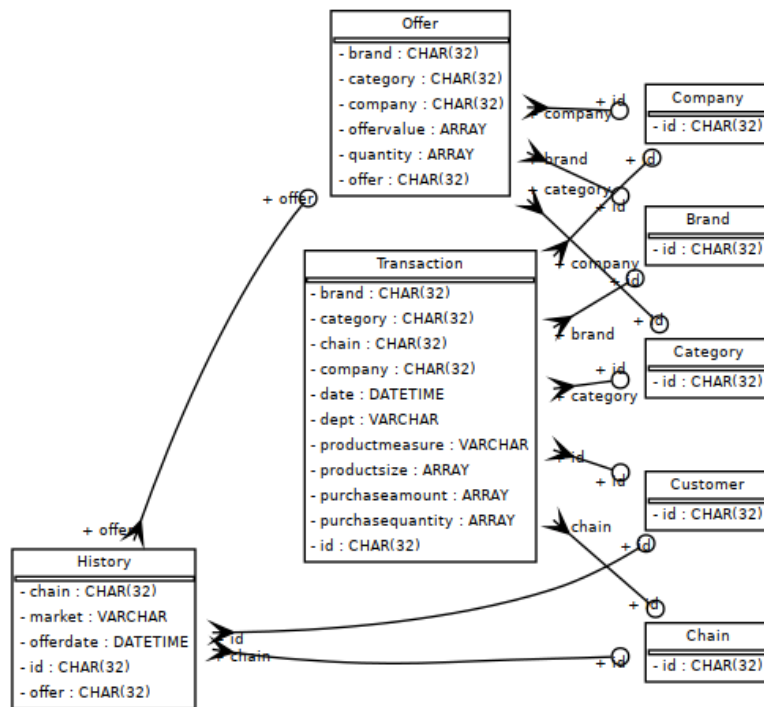
**Figure 3: Schema graph for the AVS dataset.**

*A.1.1   Task: Customer Retention Prediction (Retent.)*

```
task = dataset.get_task('repeater')
```

The task given by the dataset vendor is to predict whether a customer will be retained by the platform, i.e. `History.repeater`. Note that in the real world, there are two possible interpretations of the `repeater` column: either a given customer will be retained (hence associated with the customer only, making it an attribute of an entity), or else a given customer will repeat the same purchase promoted by the offer (hence associated with the customer and the offer, making it an attribute of a relationship). Since the vendor did not make this distinction clear, we chose the second option. Moreover, since the prediction time is likely different than the date a promotion is offered to a customer, we selected a timestamp later than the offer date.

## A.2   Outbrain Click Prediction (OB)

```
dataset = dbb.load_rdb_data('outbrain')
```

The Outbrain dataset [53] is a large relational dataset from the content discovery platform Outbrain. It contains a sample of users' page views and clicks observed on multiple publisher sites in the United States between June 14, 2016, and June 28, 2016. The dataset consists of several tables. The `Events` table provides the context information about the user events. The `Click` table shows which ads were clicked. The `Promoted` table provides details about the advertisements. The `DocumentsCategory`, `DocumentsTopic` and `DocumentsEntity` provide information about the promoted contents, as well as Outbrain's confidence in each respective relationship. In `DocumentsEntity`, an `entity_id` can represent a person, organization, or location. The rows in `DocumentsEntity` give the confidence that the given entity was referred to in the document. The dataset schema is shown in Figure 4. Table `User` is a dummy table induced from the `Pageview.uuid` and `Event.uuid` foreign key columns.

*A.2.1   Task: CTR Prediction (CTR).*

```
task = dataset.get_task('ctr')
```

The task is to predict whether a promoted content will be clicked or not, i.e. predicting `Click.clicked`.

## A.3   Diginetica Personalized E-Commerce Search Challenge (DG)
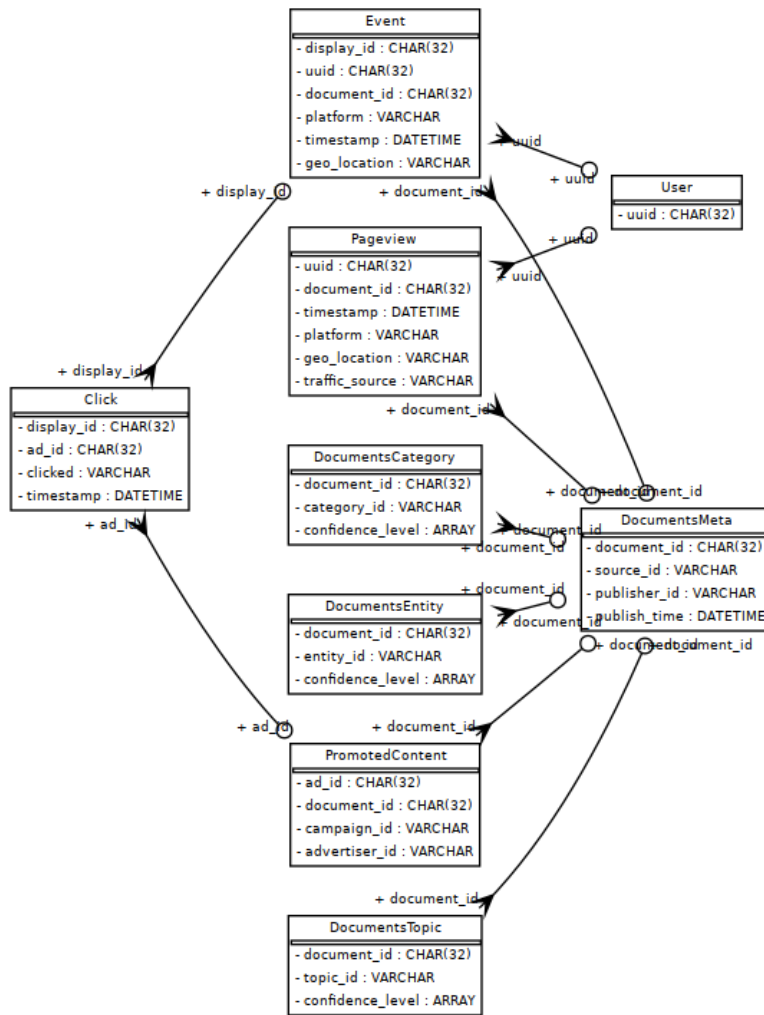
```
dataset = dbb.load_rdb_data('diginetica')
```

**Event**
- display_id : CHAR(32)
- uuid : CHAR(32)
- document_id : CHAR(32)
- platform : VARCHAR
- timestamp : DATETIME
- geo_location : VARCHAR

**User**
- uuid : CHAR(32)

**Pageview**
- uuid : CHAR(32)
- document_id : CHAR(32)
- timestamp : DATETIME
- platform : VARCHAR
- geo_location : VARCHAR
- traffic_source : VARCHAR

**Click**
- display_id : CHAR(32)
- ad_id : CHAR(32)
- clicked : VARCHAR
- timestamp : DATETIME

**DocumentsCategory**
- document_id : CHAR(32)
- category_id : VARCHAR
- confidence_level : ARRAY

**DocumentsMeta**
- document_id : CHAR(32)
- source_id : VARCHAR
- publisher_id : VARCHAR
- publish_time : DATETIME

**DocumentsEntity**
- document_id : CHAR(32)
- entity_id : VARCHAR
- confidence_level : ARRAY

**PromotedContent**
- ad_id : CHAR(32)
- document_id : CHAR(32)
- campaign_id : VARCHAR
- advertiser_id : VARCHAR

**DocumentsTopic**
- document_id : CHAR(32)
- topic_id : VARCHAR
- confidence_level : ARRAY

**Figure 4: Schema graph for the Outbrain dataset.**

The dataset `diginetica` [17] is part of the Personalized E-commerce Search Challenge and is provided by DIGINETICA and its partners. The dataset focuses on predicting the search relevance of products based on users' personal shopping, search, and browsing preferences. The `diginetica` dataset consists of several tables: `Product` contains information about the products, `Click` contains click data, etc. We conducted the following data cleaning steps from the original tables:

(1) The original data does not provide accurate session start times. Instead, the dataset provides the event date and the timeframe (in milliseconds) that each event happens relative to its session. We generate a random start time for each session and convert the relative timeframe of each event into an absolute timestamp.

(2) Product names and query search strings are represented as sequence of anonymous token IDs in the original data. We convert them into two tables `ProductNameToken` and `QuerySearchstringToken`.

(3) The original `Query` table stores the query results as a column of item ID lists. We convert that column into a separate table `QueryResult` where each entry is a triplet of `queryId`, `itemId` and `timestamp`.

Steps 2 and 3 make the database satisfy the First Normal Form (1NF) where there are only single-valued attributes [25]. Figure 5 depicts the final RDB schema. Note that `Token`, `Orders`, `Session` and `User` are dummy tables induced from the corresponding foreign keys.

*A.3.1 Task: CTR Prediction (CTR).*

```
task = dataset.get_task('ctr')
```
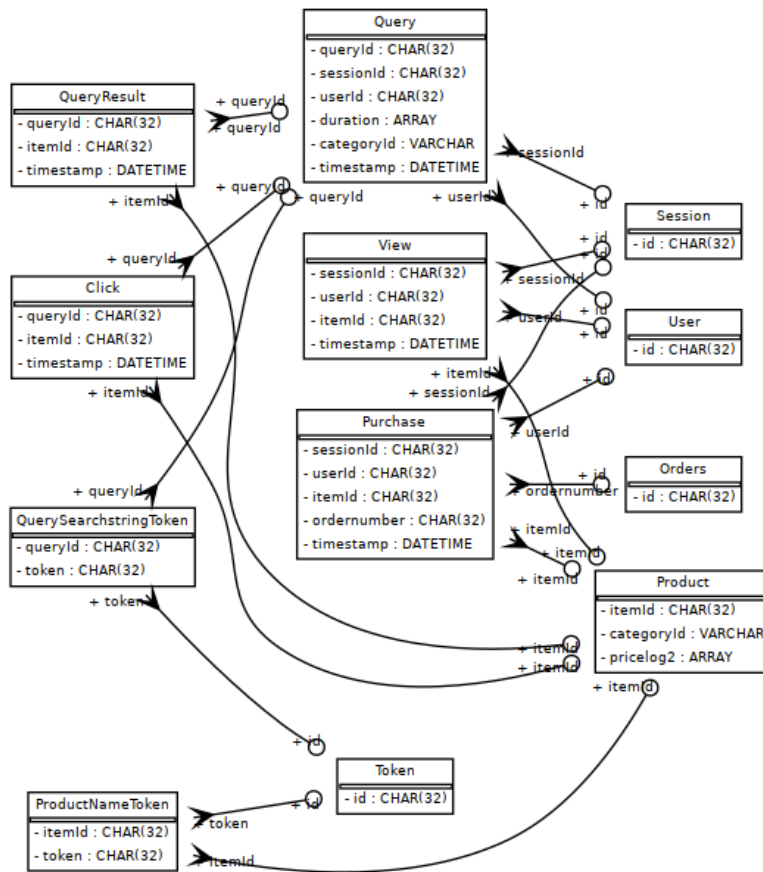
**Figure 5: Schema graph for the Diginetica dataset.**

The task is to predict whether an item will be clicked when listed by a given query, i.e., a binary classification task given a triplet of `queryId`, `itemId` and `timestamp`. Positive samples are collected from the `Click` table while negative samples are those in `QueryResult` but not in `Click`. The prediction timestamp is the first time an item is listed by a query to simulate the setting that a recommender system attempts to return the most relevant items for a query. We further down-sample the train/validation/test set to around 100K samples.

*A.3.2   Task: Product Purchase Prediction (Purch.)*

```
task = dataset.get_task('purchase')
```

This task is to predict which items will be purchased in a given session, i.e., predicting the foreign key column `Purchase.itemId`. The evaluation metric is Mean Reciprocal Rank (MRR), where the model needs to rank the positive purchase high among 100 randomly generated negative candidates.

## A.4   RetailRocket Recommender System Dataset (RR)

```
dataset = dbb.load_rdb_data('retailrocket')
```

The dataset RetailRocket [84] is a Kaggle dataset provided by the E-commerce platform RetailRocket. The recorded events represent user interactions on the website. The dataset includes several tables: `View` contains information about whether an item was added to the cart by an user. `Category` stores product category tree. The original dataset stores all item properties in the `ItemProperty` table where most of the property names and values are anonymous tokens. We extract two properties into separate tables: `ItemAvailability` marks the availability status of an item at certain timestamp; `ItemCategory` stores the category information of each item. The dataset schema is shown in Figure 6. Note that `Item`, `Visitor` are dummy tables induced from the corresponding foreign keys.

*A.4.1   Task: Conversion Rate Prediction (CVR).*
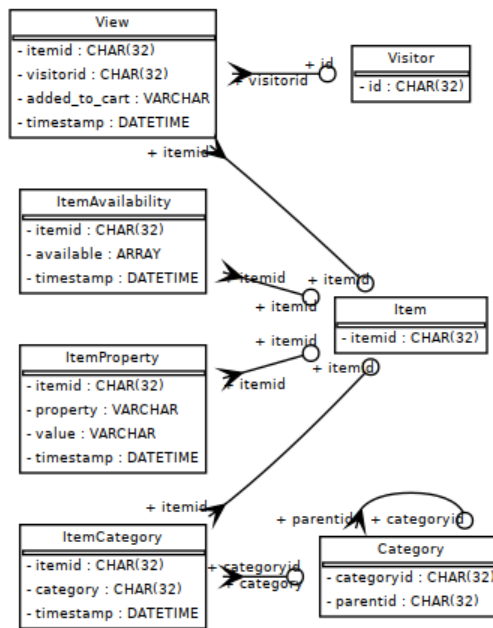
```
task = dataset.get_task('cvr')
```

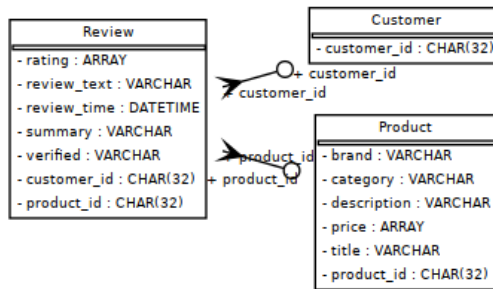**Figure 6: Schema graph for the RetailRocket dataset.**



**Figure 7: Schema graph for the Amazon Book Reviews dataset.**

The task is to classify whether an item will be added to the shopping cart by a visitor, i.e. predicting column `View.added_to_cart`. We downsampled the training/validation/testing set to contain 100K samples.

## A.5 Amazon Book Reviews (AB)

```
dataset = dbb.load_rdb_data('amazon')
```

The Amazon Review dataset [55] represents an extensive collection of product reviews on Amazon, encompassing 233 million unique reviews from approximately 20 million users. Our benchmark utilizes a 5-core subset from the Books category of the original dataset. As depicted in Figure 7, the curated dataset is organized into three tables: The `Customer` table, which catalogues unique IDs for each reviewing customer; the `Product` table, detailing each book with a unique ID, brand, category, description, price, and title; and the `Review` table, documenting each review's connection to a customer and a product, along with the review's rating, text, submission time, summary, and verification status. Spanning from June 25, 1996, to September 28, 2018, this relational database comprises 1.85M customers, 21.9M reviews, and 506K products.

### A.5.1 Task: User Churn Prediction (Churn).

```
task = dataset.get_task('churn')
```

The task is to predict whether a user will continue to engage with the platform and make any purchases in the subsequent three months, forming a binary classification challenge. We select a subset of active users—who have contributed a minimum of 10 reviews in the two years prior to the prediction timestamp—as the set for training, validation and testing.

### A.5.2 Task: Rating Prediction (Rating).

```
task = dataset.get_task('rating')
```

This task is to infer the numerical rating a user might assign to a product, i.e., a regression task on the column `Review.rating`. The prediction should rely solely on the historical review and purchase data, without access to the current review content. The model needs to identify and utilize trends in past user interactions and product engagements to accurately predict the rating, which can range from 1 to 5 stars.

We remark that in practice, when predicting the rating, columns such as `Review.review_text`, `Review.review_time`, `Review.summary` at the same row should not be used, since in real-world settings they are usually given by the customer together with the rating. Hence using these columns to predict a rating at the same row should be treated as a form of information leakage. Nevertheless, it is perfectly fine to use *historical* review texts and summaries to predict the present rating.

### A.5.3 Task: Product Purchase Prediction (Purch.)

```
task = dataset.get_task('purchase')
```

The task is to predict which product will be purchased by a given user, i.e., predicting the foreign key column `Product.product_id`. The evaluation metric is Mean Reciprocal Rank (MRR), where the model needs to rank the positive purchase high among 50 randomly generated negative candidates per positive candidate.

## A.6 StackExchange (SE)

```
dataset = dbb.load_rdb_data('stackexchange')
```

The dataset StackExchange [61] is collected from the online question-and-answer platform StackExchange. The dataset includes several tables: `Badges` includes the information of badges assigned to users; `Comments` stores comments attached to posts; `Posts`, `Tag`, `PostLink` and `PostHistory` are tables of post data; `Users` includes information of users; `Votes` indicates which posts are voted by which users. The dataset schema is shown in Figure 8. Note that prior work [23] has also relied on the same data source for benchmarking. Although we adopt the same task specification, our StackExchange dataset nonetheless remains distinct from [23] in three aspects:

(1) We retain the `Tag` table from the raw data source, which contains the linkage to an excerpt post and a Wiki post for each StackExchange tag;

(2) We expand the `Tag` attribute in the `posts` table (containing a set of tags for each post) into another table `PostTag`, thereby preserving additional structural information related to post tags;

(3) We pull more data from the raw data source, with time-stamps up until 2023-09-03; this augmentation results in 6 months more data than the one used by [23].

### A.6.1 Task: User Churn Prediction (Churn).

```
task = dataset.get_task('churn')
```

The task is to predict whether a user will make any engagement, defined as vote, comment, or post, to the site within the next 2 years starting from year 2011, 2013, 2015, 2017, and 2019 for the training set, and year 2021 for the validation set, and year 2023 for the test set. Note that in the training set, each user requires multiple predictions for different time windows. We also make sure that the prediction timestamps are always later than the user's own creation date.

### A.6.2 Task: Post Popularity Prediction (Popul.)

```
task = dataset.get_task('upvote')
```

The task is to predict whether a given post will be voted in one year since the post was created.

## A.7 Microsoft Academic Graph (MAG)

```
dataset = dbb.load_rdb_data('mag')
```

The dataset is a subset of the Microsoft Academic Graph (MAG), a knowledge graph for academic publications, venues and author information. We include this dataset to highlight the duality between graph and RDB data. We repurposed the ogbn-mag dataset from the popular Open Graph Benchmark (OGB) [35] into an RDB of multiple tables: `Paper`, `FieldOfStudy`, `Author` and `Institution` are derived from the four node types; while `Cites`, `HasTopic`, `Writes` and `AffiliatedWith` are created from the four relationships. We retained the same set of node/edge features as table attributes. Specifically, for the `Paper` table, `feat` stores the vector embeddings of each table (processed by OGB) and `label` stores which venue a paper is published at (also the prediction target of Task Venue). The final RDB schema is shown in Figure 9.
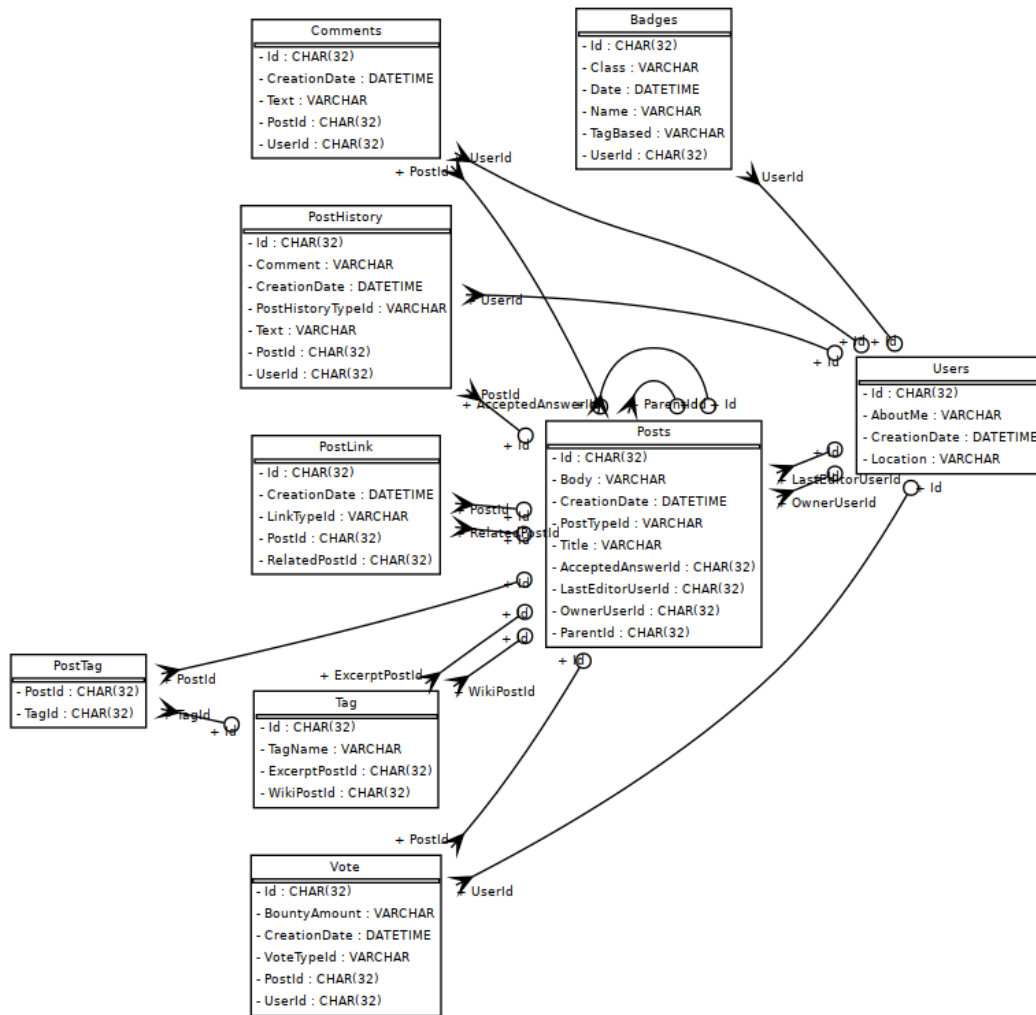
**Figure 8: Schema graph for the StackExchange dataset.**

*A.7.1 Task: Paper Venue Prediction (Venue).*

```
task = dataset.get_task('venue')
```

The task is to predict the venue (conference or journal) a paper is published at. Because the label column is included in the RDB, using the column in making the prediction of the same paper is treated as a form of information leakage, but the model is free to use the labels of connected papers. In total, there are 349 different venues, making the task a 349-class classification problem. Following the practice of `ogbn-mag`, papers are split into train, validation and test sets by their published years.

*A.7.2 Task: Citation Prediction (Cite).*

```
task = dataset.get_task('cite')
```

The task is to predict which paper will be cited, i.e., predicting the foreign key column `Cites.paper_cite`. We sampled 100K citations as positive samples, and randomly generated 100 negative candidates for each citation in the validation and test sets. The evaluation metric is Mean Reciprocal Rank (MRR).

## A.8 Seznam (SZ)

```
dataset = dbb.load_rdb_data('seznam')
```

The Seznam dataset [54] is collected from a web portal and search engine in the Czech Republic, which contains online advertisement expenditures from a customer's wallet. The dataset includes the following tables: `Client`, `Dobito` (charges), `Proběhnuto` (prepay), and
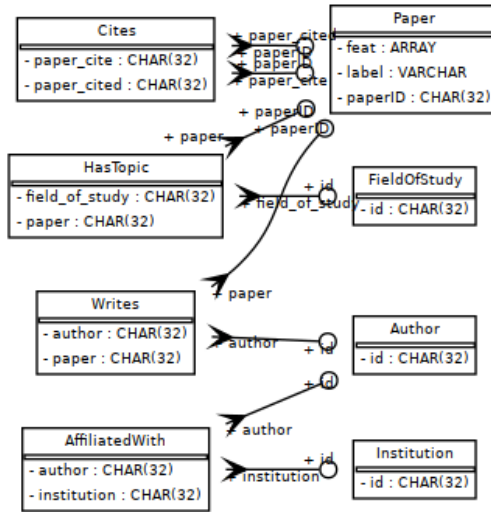
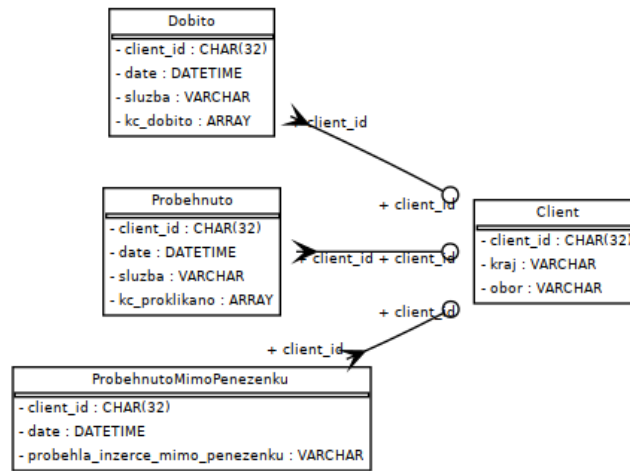**Figure 9: Schema graph for the MAG dataset.**



**Figure 10: Schema graph for the Seznam dataset.**

ProběhnutoMimoPeněženku (charges outside the wallet). The `Client` table provides the location and domain field information of the anonymized client. The `Proběhnuto` table includes information about prepayments made into a wallet in Czech currency. The `Dobito` table includes information about charges made from the wallet in Czech currency. The `ProběhnutoMimoPeněženku` table includes information about charges made in Czech currency, but not deducted from the wallet. The RDB schema is shown in Figure 10, where the timestamp column, primary keys, and foreign keys are indicated.

*A.8.1  Task: Charge Type Prediction (Charge).*

```
task = dataset.get_task('charge')
```

The task of Charge Type Prediction is to predict the type of each charge, i.e., `Dobito.sluzba`. The evaluation metric for charge prediction is accuracy among 8 classes.

*A.8.2  Task: Prepay Type Prediction (Prepay).*

```
task = dataset.get_task('prepay')
```

The task of Prepay Type Prediction is to predict the type of each prepay, i.e., `Proběhnuto.sluzba`. The evaluation metric for prepay prediction is accuracy among 8 classes.

# B ABLATIONS

## B.1 With and Without Dummy Tables

In accordance with the descriptions of each dataset and task (Appendix A), we introduce dummy tables with PK columns matched with high-cardinality columns in original RDB tables. Further details can also be found in Appendix G.4, where Row2Node+ and Row2N/E+ are introduced as two approaches (built on Row2Node and Row2N/E, respectively) that incorporate dummy tables. An example is the "Customer" table in AVS. Introducing dummy tables can enrich inter-row connections, with multiple implications. Firstly, adding dummy tables establishes new FK-PK relationships, from which DFS may obtain more features. Secondly, dummy tables introduce extra node types and edge types that can be leveraged by GNN models.

To study its impact on model performance, we choose five datasets involving dummy tables, and evaluate the performance changes for both DFS solutions and GNN solutions, with and without the presence of dummy tables. In order to remove the dummy tables, for RR and MAG, we directly drop the FK columns referring to the dummy tables. We choose this approach instead of converting these columns into categorical features in order to maintain an inductive setting. However, for the AB and AVS datasets, directly dropping these columns would result in changes to task specification. Hence, in these cases, we treat them as categorical features and drop the dummy tables.

Table 5 shows the results with and without dummy tables. For DFS with MLP model, we only report the results on AB and AVS. This is because, for the other datasets, the generated features remain unchanged regardless of the presence of dummy tables. It is evident that the performance significantly decreases without the inclusion of dummy tables, underscoring the importance of creating these tables. The situation is more complex for GNN solutions. Removing the dummy tables consistently leads to a decrease in performance for the AB, AVS, and MAG datasets. However, for results on OB datasets, there appears to be no significant difference in performance with or without dummy tables. One possible explanation is that the related columns have less importance, resulting in minimal benefits from the addition of dummy tables. Nevertheless, for certain datasets, creating dummy tables can enrich the connection relationships and effectively enhance performance.

| Dataset/Task | | AB/Rating | AVS/Retent. | OB/CTR | MAG/Venue |
|---|---|---|---|---|---|
| Prediction Type | | RA | RA | RA | EA |
| Evaluation Metric | | RMSE↓ | AUC↑ | AUC↑ | Acc.↑ |
| DFS MLP | w/ dummy | 0.9847 | 0.5690 | - | - |
| | w/o dummy | 1.0291 | 0.5469 | - | - |
| R2N R-GCN | w/ dummy | 0.9639 | 0.5653 | 0.6239 | 0.3792 |
| | w/o dummy | 1.0495 | 0.4761 | 0.6173 | 0.3762 |
| R2N R-GAT | w/ dummy | 0.9563 | 0.5637 | 0.6146 | 0.3888 |
| | w/o dummy | 1.0493 | 0.4687 | 0.6160 | 0.3771 |
| R2N/E R-GCN | w/ dummy | 0.9696 | 0.5653 | 0.6271 | 0.4671 |
| | w/o dummy | 1.0536 | 0.4708 | 0.6244 | 0.4111 |
| R2N/E R-GAT | w/ dummy | 0.9657 | 0.5638 | 0.6308 | 0.4512 |
| | w/o dummy | 1.0511 | 0.4708 | 0.6328 | 0.4071 |

**Table 5: DFS and GNN performance comparisons with or without dummy tables. Note that Row2Node+ and Row2N/E+, respectively, are used in Appendix G.4 to reference Row2Node and Row2N/E graph extraction methods augmented with dummy tables.**

## B.2 Stronger Task-Specific GNN - Neural Common Neighbors

To explore the possibility of stronger GNN architectures on a task-specific basis, we have implemented a powerful link prediction method based on the Neuron Common Neighbor (NCN) algorithm [69]. NCN is an effective approach for link prediction that includes the incorporation of common neighbor embeddings into the prediction process. This method has distinct advantages in terms of expressiveness, as common neighbors cannot be expressed in traditional message-passing GNN, and it has previously been shown to achieve state-of-the-art performance. We have conducted tests using NCN for all 8 relation attribute and foreign key tasks, and the results indicate that NCN performs quite well. Furthermore, the results demonstrate that RDB benchmark graphs still maintain certain characteristics when compared to general graph tasks, suggesting that further research on GNN can be applied to the RDB benchmark.

To restate the link prediction problem: link prediction is a widely studied issue in graph analysis, whereby a model is tasked with predicting the presence or absence of a link between two given nodes. In the context of the RDB benchmark, this problem encompasses two tasks: relation attribute prediction and foreign key prediction.

Since the original NCN method exclusively applies to static homogeneous graphs, we have expanded it to encompass temporal and heterogeneous settings. The primary process can be divided into three supplementary steps compared to simple GNN baselines: specifying common neighbor meta-paths, retrieving common neighbor embeddings in GNN layers, and appending common neighbor embeddings.

Suppose that we are given two nodes $i_0$ and $j_0$, having node types $v_{i,0}$ and $v_{j,0}$ respectively. Common neighbor meta-paths consist of two sequences of node types $[v_{i,0}, v_{i,1}, \ldots, v_{i,k_i}]$ and $[v_{j,0}, v_{j,1}, \ldots, v_{j,k_j}]$, where $v_{i,k_i} = v_{j,k_j}$ denotes the common neighbor node type, and $k_i$ and $k_j$ represent the number of common neighbor hops from $i$ and $j$. The two meta-paths specify how we discover common neighbors. We sample neighboring nodes of $i_0$ and $j_0$ with node types based on their corresponding meta-paths until we reach the final hop. Subsequently, we assess whether the sampled nodes are the same, implying that they are common neighbor nodes. Formally, we search for nodes $Common\_neighbors = \{i_{k_i} = j_{k_j}\}$ such that there exist nodes $[i_1, i_2, \ldots, i_{k_i}]$ satisfying $i_t \in \mathcal{N}_{i_{t-1}}^{v_{i,t-1},v_{i,t}}, t \in [1, \ldots, k_i]$, and $[j_1, j_2, \ldots, j_{k_j}]$ satisfying $j_t \in \mathcal{N}_{j_{t-1}}^{v_{j,t-1},v_{j,t}}, t \in [1, \ldots, k_j]$. Once we have identified the common neighbors, our goal is to retrieve their embeddings. One simple approach is to re-run GNN on the common neighbor nodes. However, we adopt a more efficient method by directly utilizing the inner GNN embeddings as the common neighbor embeddings. It is important to note that the common neighbor nodes are also sampled during the message passing, and their intermediate embeddings are also computed. Therefore, we can utilize the corresponding inner GNN embeddings $\boldsymbol{h}_{i_{k_i},\ell-k_i}^{v_{i,k_i}}$ and $\boldsymbol{h}_{j_{k_j},\ell-k_j}^{v_{j,k_j}}$.

The final embedding for prediction is derived from $MLP(\boldsymbol{h}_{i,\ell}^{v_{i,0}}||\boldsymbol{h}_{j,\ell}^{v_{j,0}}|| \sum_{i_{k_i} \in Common\_neighbors} \boldsymbol{h}_{i_{k_i},\ell-k_i}^{v_{i,k_i}} || \sum_{j_{k_j} \in Common\_neighbors} \boldsymbol{h}_{j_{k_j},\ell-k_j}^{v_{j,k_j}})$.

We have implemented NCN based on R-GCN and conducted tests on it for some Relation Attribute and Foreign Key tasks. The results are presented in Table 6 (with the best performance in boldface), where NCN outperforms R-GCN, suggesting the potential for stronger link prediction models. An intriguing phenomenon arises as the performance of NCN varies depending on the graph construction method employed, but generally, it exhibits superior performance for Row2N/E. One plausible explanation is that Row2N/E generates a denser graph, thus making it easier to discover common neighbors. Performance differences due to different graph structures are also worthwhile as a future work.

| Dataset | | AVS | OB | RR | AB | | MAG |
|---|---|---|---|---|---|---|---|
| Task | | Retent. | CTR | CTR | Rating | Purch. | Cite |
| Prediction Type | | RA | RA | RA | RA | FK | FK |
| Evaluation metric | | AUC↑ | AUC↑ | AUC↑ | RMSE↓ | MRR↑ | MRR↑ |
| R2N | R-GCN | 0.5578 | 0.6239 | 0.8470 | 0.9639 | 0.1790 | 0.6336 |
| | NCN | 0.5613 | 0.6189 | **0.8620** | 0.9750 | 0.1944 | 0.6634 |
| R2N/E | R-GCN | 0.5653 | 0.6271 | 0.8091 | 0.9696 | 0.2503 | 0.7539 |
| | NCN | **0.5658** | **0.6294** | 0.8156 | **0.9638** | **0.3071** | **0.8147** |

Table 6: Results of NCN

## B.3 Propagating Labels

In the context of classification and regression tasks, the incorporation of label information from related instances can serve as a strong signal and indicator for predicting the target. We add label information to our pipeline and consider it as a normal feature, performing label propagation in the message passing process to enhance the predictive power. With the help of the temporal neighbor sampler in our framework, any future information will be filtered out, which avoids label leakage when introducing label information.

We now examine the effectiveness of propagating labels from two angles. We select four tasks and assess the changes in performance of R-GCN under different settings. Results are presented in Table 7. Firstly, we remove the label information and only rely on features for prediction. Compared to the default setting which unitizes both label and features, this leads to a significant decrease in performance, underscoring the importance of incorporating label information. Secondly, we remove all features and retain only the label information and data structure, establishing a pure label propagation setting. As a result, in three out of four tasks, there is no noticeable decline in performance. For the RR-CTR task, although the AUC decreases by approximately 3%, the performance of only utilizing labels is still superior to that of only using features.

Based on the comparison of these three sets of results, we can conclude that related labels are often quite informative and can be a very discriminative feature. Propagating labels can therefore serve as an effective and essential approach for enhancing predictions in many practical scenarios.

## C TOOLBOX DESCRIPTION

We release the benchmarks as a Python package `dbinfer-bench`, installable via PyPI:

```
pip install dbinfer-bench
```

Besides, we also provide a toolbox `dbinfer` for running and comparing the various baselines described in the paper. We modularized the graph-centric predictive pipeline (Figure 2) into several out-of-box command-line tools:

- **Data transform and featurization command `transform`**: Loads an RDB dataset, performs a series of data transformation according to user configurations, and writes the transformed data as a new RDB dataset. The default transformations include: normalizing and canonicalizing various types of feature data (e.g., numerical, categorical, datetime columns, etc.), creating dummy tables, embedding

| Dataset/Task | AB/Rating | RR/CTR | OB/CTR | MAG/Venue |
|---|---|---|---|---|
| Prediction Type | RA | RA | RA | EA |
| Evaluation Metric | RMSE↓ | AUC↑ | AUC↑ | Acc.↑ |

| | | AB/Rating | RR/CTR | OB/CTR | MAG/Venue |
|---|---|---|---|---|---|
| | Default | 0.9639 | 0.847 | 0.6239 | 0.4338 |
| R2N | Remove label | 1.0064 | 0.7808 | 0.5233 | 0.3921 |
| | Remove features | 0.9742 | 0.817 | 0.6238 | 0.4329 |
| | Default | 0.9696 | 0.8091 | 0.6271 | 0.4895 |
| R2N/E | Remove label | 1.0057 | 0.7554 | 0.4952 | 0.4615 |
| | Remove features | 0.9566 | 0.7818 | 0.6227 | 0.5033 |

**Table 7: Results of propagating labels.**

text data into vectors, filtering redundant columns and so on. Users can plug in new data transform logic by inheriting the pre-defined interfaces such as column-wise or table-wise transformer.

- **Deep Feature Synthesis command `dfs`**: Converts an RDB dataset into a new dataset with only a single table, augmented with features produced by the Deep Feature Synthesis algorithm [41]. The command allows users to configure the search depth of the algorithm (setting depth to one gives the simple join baseline), the set of aggregators in use and the backend engine to run (FeatureTools or SQL-based engine).
- **Graph construction command `construct-graph`**: Takes an RDB dataset and produces a graph dataset using algorithms Row2Node (R2N) or Row2N/E (R2N/E).
- **Training tabular-based solution `fit-tab`**: Trains a selected tabular-based solution (e.g., MLP, DeepFM, etc.) over an RDB dataset. Running it over the original RDB dataset corresponds to the single table baseline, while the late-fusion solutions can be launched by running it over the dataset processed by DFS.
- **Training graph-based solution `fit-gml`**: Trains a selected GNN-based solution over a graph dataset.

The toolbox is designed for usability, with a philosophy of omitting unnecessary coding as much of possible. For example, researchers can freely embed some of the commands into their own pipeline, replace some steps with their own, or compose them into new solutions. Moreover, all the commands can be configured via YAML files without modification of the source code. The modularized design also lowers the complexity of fair comparisons among solutions of very different nature. For example, since the tabular-based and GNN-based solutions use the same set of featurization and data transformation steps, it establishes their comparison upon the same foundation. Similarly, researchers can also swap in/out different preprocessing steps to study their impact over the predictive architectures.

# D BASELINE IMPLEMENTATION DETAILS

## D.1 DFS-Based Models

Our implementation of DFS leverages Featuretools [22] with the following aggregators: MEAN, MAX, MIN for numeric and embedding features, MODE for categorical features, and COUNT for number of elements per aggregation (i.e. degree). For vector features, we implement custom aggregation primitives as Featuretools does not support it natively. Note that to avoid temporal leakage, for each prediction with timestamp, DFS should not aggregate information from the future beyond the given timestamp (a.k.a. *cutoff time*). Unfortunately, Featuretools does not support cutoff time very efficiently, and obtaining results on some datasets such as RetailRocket is impossible even after 60 hours on an AWS r5.24xlarge instance. While getML's FastProp [26] offers another efficient alternative to Featuretools, the propositionalization engine is implemented in C++ and extending it with custom primitives to support aggregation of embeddings is not trivial. So we developed another solution that translates feature aggregation generated by DFS into SQL, which are then executed by DuckDB.[7]

The predictive models we choose are MLP, DeepFM [31], FT-Transformer [30] and XGBoost [8]. We run XGBoost by invoking TabularPredictor from AutoGluon [21], restricting the candidate models to XGBoost only. For MLP, DeepFM and FT-Transformer, we first project each column into a fixed-length representations using a linear layer, treating categorical variables as one-hot vectors. For MLP, we concatenate all fixed-length representations into a single vector as input. For DeepFM and FT-Transformer, we treat each representation as separate field. The hyperparameter grid is shown in Table 8.

## D.2 GNN Models

We choose **R-GCN** [57], **R-GAT** [6], **HGT** [37], and **R-PNA** (extending [13] to heterogeneous graphs) as our baselines, with the hyperparameter grid shown in Table 9. The choice of PNA is due to its multitude of aggregators, resembling DFS.

However, the models do not naturally account for edge features. Since Row2N/E could convert some tables to edges and therefore some columns to edge features, we must also extend the aforementioned four models to use edge feature inputs accordingly. The high-level idea is

---

[7]https://duckdb.org/

| Hyperparameter | Values |
|---|---|
| Fixed-length representation size | {8,16} |
| Hidden dimension size | {128,256} |
| Dropout | {0, 0.1, 0.3, 0.5} |
| Number of layers | {2, 3, 4} |
| FT-Transformer attention heads | 8 |

Table 8: Hyperparameter grid for DFS-based models.

| Hyperparameter | Values |
|---|---|
| Fixed-length representation size | {8,16} |
| GNN layers | {2,3} |
| Neighbor sampling fanout | {5, 10, 20} |
| Hidden dimension size | {128, 256} |
| Dropout | {0, 0.1, 0.3, 0.5} |
| Number of predictor MLP layers | {2, 3, 4} |
| GAT/HGT attention heads | {4, 8} |
| PNA aggregators | Mean, Min, Max |

Table 9: Hyperparameter grid for GNN-based models.

to (1) project the features on each edge $e$ into a fixed-length representation $x_e$, (2) during message passing from a node $u$ to a node $v$ along edge $e$, $x_e$ is also sent along with $u$'s own representation to $v$. The following describes the mathematical details.

To facilitate discussion, denote $u, v$ as nodes, and triplet $(u, e, v)$ as an edge connecting from $u$ to $v$ with a unique identifier $e$. Moreover, denote $t(v)$ as the node type of $v$, and $\tau(e)$ as the edge type of $e$.

### D.2.1 R-GCN and R-PNA. R-GCN can be expressed as

$$h_v^{(l)} = \sigma \left( \mathbf{W}_{t(v)}^{(l)} h_v^{(l-1)} + \sum_{(u,e,v) \in \mathcal{N}(v)} c_e \mathbf{W}_{\tau(e)}^{(l)} h_u^{(l-1)} \right) \tag{8}$$

where $\mathcal{N}(v)$ represents the edges going towards $v$ and $c_e$ is some normalization constant associated with edge $e$. To account for edge features $x_e$, we extend it to (changes to the previous equation highlighted in blue)

$$h_v^{(l)} = \sigma \left( \mathbf{W}_{t(v)}^{(l)} h_v^{(l-1)} + \sum_{(u,e,v) \in \mathcal{N}(v)} c_e \mathbf{W}_{\tau(e)}^{(l)} \left[ h_u^{(l-1)} \| x_e \right] \right) \tag{9}$$

We extend R-PNA to handle $x_e$ in a similar fashion.

### D.2.2 R-GAT. R-GAT's formulation is similar to R-GCN's, except that one replaces $c_e$ with a parametrized attention function $\alpha$:

$$h_v^{(l)} = \sigma \left( \mathbf{W}_{t(v)}^{(l)} h_v^{(l-1)} + \sum_{(u,e,v) \in \mathcal{N}(v)} \alpha_{\tau(e)} \left( h_v^{(l-1)}, h_u^{(l-1)}; \mathbf{V}_{\tau(e)}^{(l)} \right) \mathbf{W}_{\tau(e)}^{(l)} h_u^{(l-1)} \right)$$

$$\alpha_{\tau(e)} \left( h_v^{(l-1)}, h_u^{(l-1)}; \mathbf{V}_{\tau(e)}^{(l)} \right) = softmax_u \left[ a_{\tau(e)} \left( h_v^{(l-1)}, h_u^{(l-1)}; \mathbf{V}_{\tau(e)}^{(l)} \right) \right] \tag{10}$$

$$a_{\tau(e)} \left( h_v^{(l-1)}, h_u^{(l-1)}; \mathbf{V}_{\tau(e)}^{(l)} \right) = LeakyReLU \left( \mathbf{V}_{\tau(e)}^{(l)} [h_u^{(l-1)} \| h_v^{(l-1)}] \right)$$

We extend the equations above to

$$h_v^{(l)} = \sigma \left( \mathbf{W}_{t(v)}^{(l)} h_v^{(l-1)} + \sum_{(u,e,v) \in \mathcal{N}(v)} \alpha_{\tau(e)} \left( h_v^{(l-1)}, h_u^{(l-1)}, x_e; \mathbf{V}_{\tau(e)}^{(l)} \right) \mathbf{W}_{\tau(e)}^{(l)} \left[ h_u^{(l-1)} \| x_e \right] \right)$$

$$\alpha_{\tau(e)} \left( h_v^{(l-1)}, h_u^{(l-1)}, x_e; \mathbf{V}_{\tau(e)}^{(l)} \right) = softmax_u \left[ a_{\tau(e)} \left( h_v^{(l-1)}, h_u^{(l-1)}, x_e; \mathbf{V}_{\tau(e)}^{(l)} \right) \right] \tag{11}$$

$$a_{\tau(e)} \left( h_v^{(l-1)}, h_u^{(l-1)}, x_e; \mathbf{V}_{\tau(e)}^{(l)} \right) = LeakyReLU \left( \mathbf{V}_{\tau(e)}^{(l)} \left[ h_u^{(l-1)} \| h_v^{(l-1)} \| x_e \right] \right)$$

*D.2.3   HGT.* For HGT, we make two changes to make it account for $x_e$: (1) change the *MSG-head$^i$* function in Equation 4 from [37] to additionally concatenate the edge representation $x_e$ before linear projection, (2) in the computation of *ATT-head$^i$* in Equation 3 from [37], add to $W_{\phi(e)}^{ATT}$ a matrix that is a parametrized linear projection of $x_e$.

# E   FURTHER DETAILS REGARDING PRIOR CANDIDATE RDB BENCHMARKS

Since OpenML [64] contains exclusively single table datasets, and OGB [34, 36], HGB [51], and TGB [38] contains exclusively graphs processed from raw data, we only discuss RelBench, RDBench and CTU Prague Relational Learning Repository in detail.

## E.1   RelBench

RelBench [23] is among a series of contemporary works that highlights the need to transition from graph machine learning to RDBs. As of the time of this writing, RelBench has two datasets: Amazon Book Reviews and StackExchange. Two tasks are designated for Amazon Book Reviews: (1) predicting whether a user will churn beyond a certain timestamp, (2) predicting a user's long term value beyond a certain timestamp. The two tasks designated for StackExchange are: (1) predicting whether a user will churn beyond a certain timestamp, (2) predict the number of upvotes a post will receive at a given time window. In all tasks, RelBench designates multiple time windows for every entity to make prediction, effectively making all tasks a time series forecasting task. Presently though, [23] does not come with experimental results.

## E.2   RDBench and CTU Prague Relational Learning Repository

CTU Prague Relational Learning Repository (CRLR) [54] is a collection of 62 real-world and 21 synthetic relational databases. Each relational database comes with one task. However, most of the data are not purposed for a modern RDB machine learning benchmark for the following reasons:

(1) Most of the data are dated well before the big data era: only 15 real-world datasets have more than 10,000 labeled instances.
(2) The RDB schema is degenerate in that a one-hop outer join can produce a table with no loss of information. An example is the `Airline` dataset.[8]
(3) Some tasks are sufficiently easy such that our baselines already achieve 100% accuracy, which compromises our purpose to of using such benchmarks for further advancing RDB machine learning research. The `Accidents` [9] dataset is one such representative example. On the other hand, some tasks have either a very high degree of difficulty or too many noisy or irrelevant features, such that DFS and GNN-based solutions did not show any difference relative to single-table baselines.
(4) The business justification of the tasks are not as good as the ones typically found in data science competitions. An example is the `IMDB` [10] dataset, where the task is to predict an actor's gender.

Prior work RDBench [79] attempted to address the problems 1 and 2 listed above by handpicking a subset, and specifying multiple tasks for each dataset. However, they did not attempt to address problem 3, and some tasks are still too easy, reaching 100% accuracy or 0 regression error.

# F   SYSTEM RUNNING TIME

Table 10 records the running time of `dbinfer` for both DFS-based and GML-based solutions. For GML, we measure the time to train a GraphSAGE model for an epoch, including forward, backward and weight update. The DFS time is measured using our SQL-based engine which is typically 10x-1000x faster than using FeatureTools [22].

# G   FURTHER DETAILS ON THE CONVERSION OF RDBS TO GRAPHS

A *heterogeneous graph* $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ [63] is defined by sets of node types $V$ and edge types $E$ such that $\mathcal{V} = \bigcup_{v \in V} \mathcal{V}^v$ and $\mathcal{E} = \bigcup_{e \in E} \mathcal{E}^e$, where $\mathcal{V}^v$ references a set of $n^v = |\mathcal{V}^v|$ nodes of type $v$, while $\mathcal{E}^e$ indicates a set of $m^e = |\mathcal{E}^e|$ edges of type $e$. Both nodes and edges can have associated features, denoted $x_i^v$ and $z_j^e$ for node $i$ of type $v$ and edge $j$ of type $e$ respectively. Additionally, if we allow for typed edges linking together arbitrary numbers of nodes possibly greater than two, which defines a so-called hyperedge, then $\mathcal{G}$ generalizes to a heterogeneous *hypergraph* [5, 62], a perspective that will provide useful context below. And finally, for a dynamic graph $\mathcal{G}(s)$, all of the above entities can be generalized to depend on a state variable $s$ as before. We next consider two practical approaches for converting an RDB into a heterogeneous graph (or possibly hypergraph). The motivation here is straightforward: *even if we believe that graphs are a sensible route for pre-processing RDB data, we should not prematurely commit to only one graph extraction procedure.*

## G.1   Row2Node

Perhaps the most natural and intuitive way to convert an RDB $\mathcal{D}$ to a heterogeneous graph $\mathcal{G}$ is to simply treat each row as a node, each table as a node type, and each FK-PK pair as a directed edge. Additionally, non-FK/PK column values are converted to node features assigned to the respective rows. Per this construction, row $T_{i:}^k$ defines a node of type $v = k$. Similarly, if $T_{:j}^k$ represents an FK column of table $k$

---

| Dataset / Task | GML Epoch Time | DFS Time (featuretools) | DFS Time (ours) |
|---|---|---|---|
| AVS / Retent. | 24.2 | 647.4 | 175.8 |
| OB (downsampled) / CTR | 5.74 | 16682.43 | 9.78 |
| DN / CTR | 45.7 | > 10 hours | 287 |
| DN / Purch. | 15.7 | > 10 hours | 417 |
| RR / CTR | 12.7 | > 10 hours | 1372 |
| AB / Churn | 134 | > 10 hours | 25402 |
| AB / Rating | 31.9 | > 10 hours | 3746 |
| AB / Purch. | 128 | > 10 hours | 3773 |
| SE / Churn | 15.2 | > 10 hours | 2802 |
| SE / Popul. | 62.7 | > 10 hours | 2241 |
| MAG / Venue | 26.2 | 6249 | 3812 |
| MAG / Cite | 40.6 | 35 | 99 |
| SZ / Charge | 24.2 | 121.4 | 12.6 |
| SZ / Prepay | 60.9 | 298.2 | 16.2 |

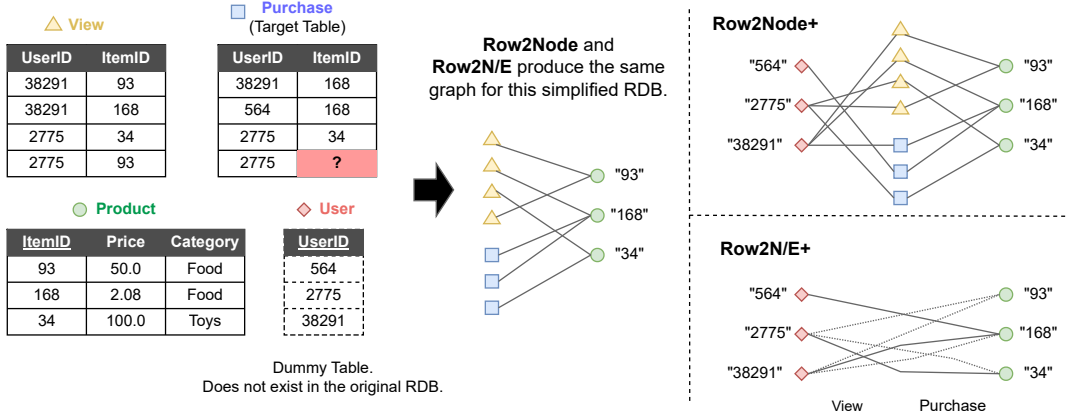**Table 10: Running Time of the GML and DFS pipeline (sec).**



**Figure 11: Illustration of Row2Node, Row2N/E and their extended version that includes dummy table.**

that references $T_{:j'}^{k'}$, the PK column $j'$ of table $k'$, then there exists an edge of type $e = kk'$ between the corresponding nodes whenever $T_{ij}^{k} \rightarrow T_{i'j'}^{k'}$ for row indices $i$ and $i'$. We will refer to this graph composition as *Row2Node* for convenience. As a relatively straightforward procedure for extracting graphs from RDBs, Row2Node was proposed in [15] with ongoing application by others [23, 77, 79]. Figure 11 visualizes the graph constructed by Row2Node from an RDB of three tables: "View", "Purchase" and "Product".

## G.2 Row2N/E

As an alternative to Row2Node, we may relax the restriction that every row must be exclusively converted to a node. Instead, for tables with two FK columns, i.e., $T_{:j}^{k}$ and $T_{:j'}^{k}$ are both FKs with $j \neq j'$, we convert each row to an edge connecting the corresponding rows being indexed by the FK pair. More concretely, each such $\{T_{ij}^{k}, T_{ij'}^{k}\}$ pair defines an edge of type $e = k'k''$ between rows of tables $T^{k'}$ and $T^{k''}$ as pointed to by $T_{:j}^{k}$ and $T_{:j'}^{k}$ respectively. The remaining columns of $T^{k}$ are designated as edge features. Overall, the intuition here is simply that tables with multiple FKs can be treated as though they were natively a tabular representation of edges.

Additionally, to ensure edges exclusively connect to nodes instead of edges as required in forming a canonical graph, we only convert rows as described above to edges if table $T^{k}$ has no PK column. If it were to have both two FK columns *and* a PK column, then a referencing FK in yet another table could lead to edge-edge connections which are disallowed by convention.[11] Additionally, we may expand this procedure to generate arbitrary hyperedges [5] by analogously handling tables with three or more FK columns. We henceforth refer to this conversion procedure as *Row2N/E* (short for Row-to-Node-or-Edge), as each row is now selectively treated as either a node or edge/hyperedge depending

---

[11]In other words, if a table has a PK, then edges from other tables may point to each row; however, if there are also two FK columns, and each row is also converted to an edge, the result would be disallowed edge-edge connections.

on the presence of multiple FKs. And if no table within $\mathcal{D}$ has multiple FKs (along with no PK column), then Row2Node and Row2N/E are equivalent. Figure 11 illustrates such a case since both "View" and "Purchases" contain only one foreign key (i.e., "ItemID"). We will see their difference later when more foreign keys are introduced (such as by adding dummy tables).

## G.3 Comparative Analysis

In general there is no ground-truth "correct" graph that can be extracted from an RDB, and hence, no *a priori* gold standard under which we might conclusively prefer Row2Node or Row2N/E, or even something else altogether (e.g., see Section G.4). Nonetheless, there does exist one relevant sanity check with the potential to influence our preferences here. This check relates to a precise form of cycle consistency as follows.

Suppose we are given an initial graph $\mathcal{G}$ as well as a general mapping $\mathcal{A}$ that converts this graph to an RDB via $\mathcal{D} = \mathcal{A}(\mathcal{G})$. We may then apply either Row2Node or Row2N/E to $\mathcal{D}$ and determine if we recover the original $\mathcal{G}$. Ideally, we would like $\mathcal{A}$ to output RDBs in some type of canonical form; otherwise achieving the aforementioned cycle consistency may either be impossible or underdetermined via Row2Node or Row2N/E. As a trivial hypothetical example, the case where $\mathcal{A}$ simply converts every edge of $\mathcal{G}$ to a row within a single table specifying the head node, tail node, relation type, and any associated node/edge features. While the graph is fully specified, it also follows that $K = 1$, there are no FKs pointing to other tables, and both Row2Node and Row2N/E will degenerate to a disconnected graph with a single node type. Hence any meaningful cycle-consistency check must be predicated on a principled choice for $\mathcal{A}$ that precludes such specious possibilities.

Fortunately though, there exist well-established methods for normalizing RDBs into canonical forms [25] that naturally filter out these types of degeneracy and can be repurposed to actualize a reasonable idempotency check.

PROPOSITION 1. *Let $\mathcal{G}$ denote a heterogeneous graph and $\mathcal{A}$ a mapping that converts $\mathcal{G}$ to a degenerate single table RDB as described above. Furthermore, let Norm denote an operator that normalizes an RDB according to the first through forth database normal forms.[12] Then Row2Node and Row2N/E as specified in Sections G.1 and G.2 are such that*

$$
\begin{aligned}
\mathcal{G} &\neq Row2Node\,[\,Norm\,(\,\mathcal{A}[\mathcal{G}]\,)\,] \\
\mathcal{G} &= Row2N/E\,[\,Norm\,(\,\mathcal{A}[\mathcal{G}]\,)\,]\,.
\end{aligned}
\tag{12}
$$

Informally, Proposition 1 demonstrates that Row2N/E has an advantage in terms of recovering a ground-truth graph that has been converted to a properly normalized/standardized RDB as quantified by well-studied database normal forms. Of course we cannot necessarily infer from this that Row2N/E is broadly preferable. Even so, this result is one noteworthy attribute worthy of consideration. Beyond this, we remark that Row2Node and Row2N/E can also be related through the notion of star-graph expansions of hypergraphs [82] via

$$
\begin{aligned}
Row2Node\,[\,Norm\,(\,\mathcal{A}[\mathcal{G}]\,)\,] \\
= Star\,(\,Row2N/E\,[\,Norm\,(\,\mathcal{A}[\mathcal{G}]\,)\,]\,)\,.
\end{aligned}
\tag{13}
$$

In this expression Star($\cdot$) produces the star-graph expansion of an arbitrary input hypergraph $\mathcal{G}$. Although star-graph expansions have limitations [72], they are nonetheless widely used to process hypergraphs [1, 70]. That being said, as will be discussed in Section 3.3 and empirically tested in Section 6, predictive baseline models built upon Row2Node and Row2N/E need not perform the same even under the restrictive setting of input RDBs constructed as $\mathcal{D} = Norm\,(\,\mathcal{A}[\mathcal{G}]\,)$; likewise for more diverse regimes/RDBs where generally Row2Node$[\mathcal{D}] \neq Star\,(\,Row2N/E\,[\mathcal{D}]\,)$.

## G.4 Extension to Row2Node+ and Row2N/E+

Thus far we have assumed that when converting an RDB to a graph, edges are exclusively formed by *known* PK-FK pairs. But practical use cases (as reflected in the benchmarks we will introduce later) sometimes warrant the invocation of another less obvious type of edge formation. Specifically, although not explicitly labeled as such, within an RDB there may exist one or more table columns with the signature of an FK (e.g., elements are a high cardinality categorical index or related), but with *no associated PK column in another table*. Moreover, the RDB may also contain a second column with elements drawn from the same high-cardinality domain; this additional column may reside in either the same or a different table as the original. Together these *pseudo FK* columns can be converted to actual FKs by introducing a new dummy table, with just a single column treated as a PK, defined by the unique corresponding elements of the pseudo FKs. In this way extracted graphs have additional pathways for sharing information across or within the original tables by passing through nodes associated with the new dummy table.

Note that this conversion of pseudo FK pairs (or the natural extension to arbitrary FK tuples) can be integrated within either Row2Node or Row2N/E, and we henceforth refer to these variants as *Row2Node+* and *Row2N/E+* respectively. In the example depicted in Figure 11, the column "UserID" is a pseudo FK, which if converted to an actual FK, gives birth to an additional *dummy* "User" table. Consequently, Row2N/E+ further treats entries in the "View" and "Purchase" table as edges, resulting in different graph constructed than Row2Node+. Please see Appendix B.1 for ablations using both Row2Node+ and Row2N/E+.

In practice there is no strict objective standard for when to convert pseudo FK tuples into dummy tables and new FK-PK pairs as described above; however, the process of finding candidates for such a conversion closely mirrors the notion of Joinable Table Discovery (JTD) [19].

---

[12]We remark that the first four normal forms are the most common normalizations used in practice [52]. For further details on database normalization and normal form definitions, we refer the reader to [25].

Although JTD is often applied on data lakes with a vast number of tables, the same logic can also be applied on RDBs to discover such pseudo FK tuples, if one treats RDB as a "small" data lake.

## G.5 Proof of Proposition 1

Given a heterogeneous attributed graph $\mathcal{G}$, we can always construct a single table $\bar{T} = \mathcal{A}(\mathcal{G})$, via an injective mapping $\mathcal{A}$, such that the $i$-th row satisfies

$$\bar{T}_{i:} = [u, w, v_u, v_w, x_u, x_w, e_i, z_i], \tag{14}$$

where $u$ and $w$ represent the head and tail node indeces of edge $i \in \mathcal{E}$. Moreover, with some abuse of notation $\{v_u, v_w\}$ and $\{x_u, x_w\}$ represent the corresponding node types and node features, respectively. Meanwhile, $e_i$ and $z_i$ indicate the relation type and (optional) feature of edge $i$. Additionally, as $\mathcal{G}$ is a heterogeneous graph (as opposed to multi-graph), each triplet $\{u, w, e_i\}$ of head node, tail node, and relation type is unique and serves as a candidate key for the table. (Note that node and edge features, as well as node types, cannot contribute to a candidate key as we make no assumptions on their values, e.g., they could all in principle be equal or non-distinguishing.)

From here, by assumption $\bar{T}$ will have unique rows such that it satisfies the criteria for an unnormalized form (UNF), i.e., no duplicated rows. Next, provided we treat each node and edge feature as a single entity, then the first normal form (1NF) is satisfied (if we were to treat each feature as a set or nested record, technically it would not, but for our purposes this distinction is inconsequential). Proceeding further, to address the second normal form (2NF) we examine all non-candidate key attributes to determine which are dependent on the entire candidate key and which are not. Clearly $z_i$ does in fact depend on the entire candidate key, and so it satisfies 2NF. Notably though, $x_u$ and $v_u$ only depend on $u$, while $x_w$ and $v_w$ only depend on $w$; in neither case is there dependency on the entire candidate key. Hence to satisfy 2NF, we must form a second table to record non-duplicated records of node features and node types, and remove these attributes from $\bar{T}$. Hence rows of $\bar{T}$ simplify to

$$\bar{T}_{i:} = [u, w, e_i, z_i] \tag{15}$$

and we introduce the node attribute table $\bar{T}^{node}$ with row $u$ given by

$$\bar{T}^{node}_{u:} = [u, v_u, x_u]. \tag{16}$$

Both $\bar{T}^{node}$ and $\bar{T}$ now satisfy 2NF, with $u$ serving as the primary key for the former, while $u$ and $w$ now independently serve as foreign keys for the latter. Next, as there are no transitive functional dependencies, nor multivalued dependencies, the third normal form 3NF and forth normal form (4NF) are trivially satisfied. We may therefore conclude that our new tables satisfy

$$\{\bar{T}, \bar{T}^{node}\} = \text{Norm} \left( \mathcal{A}[\mathcal{G}] \right) \tag{17}$$

per our previous definitions.

From here we observe that Row2Node will introduce new nodes associated with each row of *both* $\bar{T}$ and $\bar{T}^{node}$, with the former *not* present in the original $\mathcal{G}$. From this it follows that

$$\mathcal{G} \neq \text{Row2Node} \left[ \text{Norm} \left( \mathcal{A}[\mathcal{G}] \right) \right]. \tag{18}$$

As for Row2N/E, because of the newly-introduced FK-PK relationship, it naturally follows that

$$\mathcal{G} = \text{Row2N/E} \left[ \text{Norm} \left( \mathcal{A}[\mathcal{G}] \right) \right], \tag{19}$$

completing the proof.