Position: Relational Deep Learning - Graph Representation Learning on Relational Databases

Matthias Fey [*1] **Weihua Hu** [*1] **Kexin Huang** [*2] **Jan Eric Lenssen** [*1 3] **Rishabh Ranjan** [*2] **Joshua Robinson** [*2]
Rex Ying [4] **Jiaxuan You** [5] **Jure Leskovec** [2 1]

Abstract

Much of the world's most valued data is stored in relational databases and data warehouses, where the data is organized into tables connected by primary-foreign key relations. However, building machine learning models using this data is both challenging and time consuming because no ML algorithm can directly learn from multiple connected tables. Current approaches can only learn from a single table, so data must first be manually joined and aggregated into this format, the laborious process known as feature engineering. This position paper introduces *Relational Deep Learning (RDL)*, a blueprint for end-to-end learning on relational databases. The key is to represent relational databases as temporal, heterogeneous graphs, with a node for each row in each table, and edges specified by primary-foreign key links. Graph Neural Networks then learn representations that leverage all input data, without any manual feature engineering. We also introduce RELBENCH, and benchmark and testing suite, demonstrating strong initial results. Overall, we define a new research area that generalizes graph machine learning and broadens its applicability.

1. Introduction

The information age is driven by data stored in ever-growing relational databases and data warehouses that have come to underpin nearly all technology stacks. Relational databases store information in multiple tables, with entities/rows in different tables connected using primary and foreign keys and managed using powerful query languages such as SQL (Codd, 1970; Chamberlin & Boyce, 1974). For this reason, they lie at the foundation of today's large information systems, including e-commerce, social media, banking systems, healthcare, manufacturing, and open-source scientific repositories (Johnson et al., 2016; PubMed, 1996).

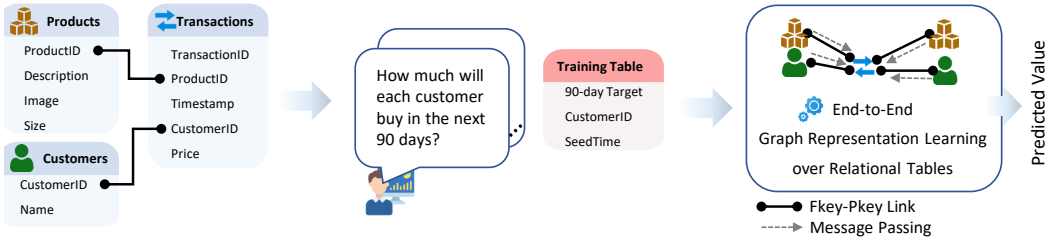
Many predictive problems over relational databases have significant implications for human decision making. A hospital wants to predict the risk of discharging a patient; an e-commerce company wishes to forecast future sales of each of their products; a telecommunications provider wants to predict which customers will churn; and a music streaming platform must decide which songs to recommend to a user. Behind each of these tasks is a rich relational schema, and many machine learning models are built using this data (Kaggle, 2022).

However, existing learning paradigms, notably tabular learning, cannot directly learn across multiple tables. Instead, a manual feature engineering step is first taken, where domain knowledge is used to manually join and aggregate tables into a single table format, where each column represents a different feature. To illustrate this, consider a simple e-commerce schema (Fig. 1) of three tables: CUSTOMERS, TRANSACTIONS and PRODUCTS, where CUSTOMERS and PRODUCTS are linked to TRANSACTIONS, and the task is to predict if a customer is going to complete any transactions in the next k days. In this case, a data scientist would aggregate information from the TRANSACTIONS table to make new features for the CUSTOMERS table such as: "number of purchases of a given customer in the last 30 days", "number of purchases in the last 14 days", "number of purchases on a Sunday", "number of purchases on a Monday", and so on. The new features for each customer are stored in a single table, ready for tabular machine learning.

There are many issues with with the above approach: (1) it is a manual, slow and labor intensive process; (2) feature choices are likely highly-suboptimal; (3) only a small fraction of the overall space of possible features can be manually explored; (4) flattening data into a single table

[*]Equal contribution [1]Kumo.AI [2]Stanford University [3]Max Planck Institute for Informatics [4]Yale University [5]University of Illinois at Urbana-Champaign. Correspondence to: Jure Leskovec <jure@cs.stanford.edu>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.



(a) Relational Database **(b)** Define Tasks **(c)** Relational Deep Learning

Figure 1. **Relational Deep Learning solves predictive tasks on relational data with end-to-end learnable models.** There are three main steps. (a) A relational database with multiple tables connected by primary-foreign keys is given. (b) A predictive task is specified and added to the database by introducing an additional training table. (c) Relational data is transformed into its *Relational Entity Graph*, and a Graph Neural Network is trained over the graph with the supervision provided by the training table. The predictive task can be node level (as in this illustration), link level (pairs of nodes), or higher-order.

aggregates data into lower-granularity features, foregoing valuable fine-grain signal; (5) features are often temporal and must be recomputed frequently, adding computational cost and risking time leakage bugs (Kapoor & Narayanan, 2023); (6) features may become obsolete over time and new features have to be manually reinvented.

Many domains have been in a similar position, including pre-deep-learning computer vision, where hand-chosen convolutional filters (*e.g.*, Gabor) were used to extract features, followed by models such as SVMs or nearest neighbor search (Varma & Zisserman, 2005). In the case of computer vision (He et al., 2016; Russakovsky et al., 2015), as in many other domains, the key was to move from manual feature engineering and handcrafted systems to fully data-driven, end-to-end representation learning systems. For relational data, this transition has not yet occurred.

Here we introduce *Relational Deep Learning (RDL)*, a blueprint for an end-to-end deep learning paradigm for relational databses (Fig. 1). Through end-to-end representation learning, RDL fully utilizes the rich predictive signals available in relational tables. The core of RDL is to represent a relational database as a temporal, heterogeneous *Relational Entity Graph*, where each row defines a node, columns define node features, and primary-foreign key links define edges. Graph Neural Networks (GNNs) (Gilmer et al., 2017; Hamilton et al., 2017) can then be applied to build end-to-end data-driven predictive models.

All in all, RDL has four main steps (Fig. 2): Given a predictive machine learning task, (1) A *training table* containing supervision labels automatically computed based on historic data in the relational database, (2) entity-level features are extracted and encoded from each row in each table to serve as node features, (3) node representations are learned through an inter-entity message-passing GNN that exchanges information between entities with primary-foreign key links, (4) a task-specific model head produces predictions, and errors backpropogated through the network.

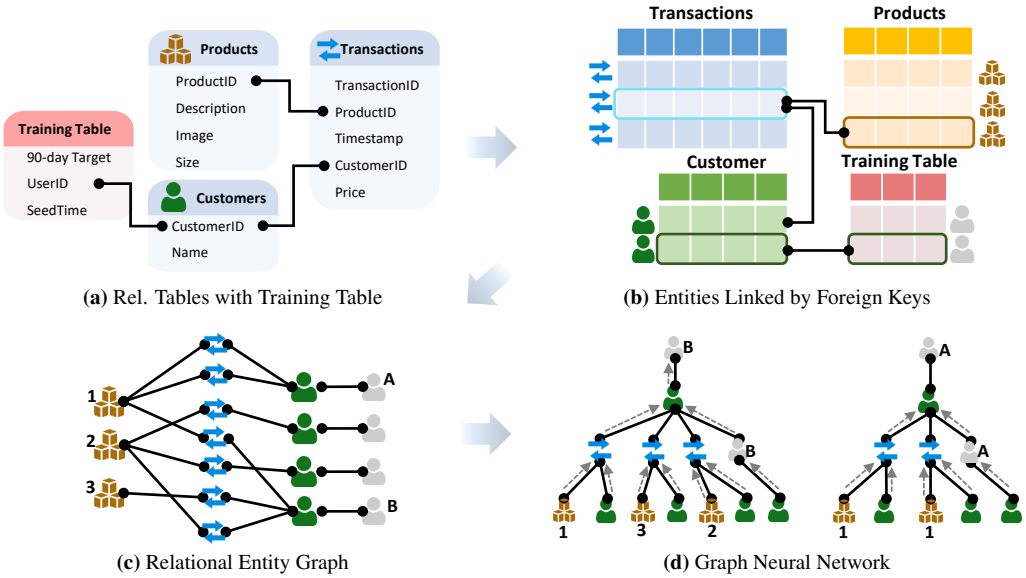
Crucially, RDL models natively integrate database temporality by only allowing entities to receive messages from other entities with earlier timestamps. This ensures that the learned representation is automatically updated with new data if a GNN forward pass is run at a later time, and prevents information leakage and time travel bugs.

This paper advocates for graph-based deep learning on relational databases. This paper lays the ground for future with the following main contributions:

- **Blueprint.** *Relational Deep Learning*, an end-to-end learnable approach that ultilizes the predictive signals available in relational data, and supports temporal predictions.
- **Prototype Implementation.** RELBENCH, an open-source implementation of RDL based on PyTorch Frame for tabular learning (Hu et al., 2024) and PyTorch Geometric for graph neural networks (Fey & Lenssen, 2019). Preliminary testing suggests significant improvements over single-table methods such as XGBoost.
- **Research Opportunities.** Outlining a new research program for Relational Deep Learning, including multi-task learning, new GNN architectures, multi-hop learning, and more.

2. Predictive Tasks on Relational Databases

This section outlines our problem scope: predictive tasks on relational tables. We define relational tables, and how to specify predictive tasks. This section focuses exclusively on the structure of data and tasks, laying the groundwork for Section 3, which presents our GNN-based approach.



(a) Rel. Tables with Training Table

(b) Entities Linked by Foreign Keys

(c) Relational Entity Graph

(d) Graph Neural Network

Figure 2. **Relational Deep Learning Pipeline. (a)** Given relational tables and a predictive task, a training table, containing supervised label information, is constructed and attached to the entity table(s). **(b)** Relational tables contain individual entities that are linked by foreign-primary key relations. **(c)** Relational data can be viewed as a single *Relational Entity graph*, which has a node for each entity, and edges given by primary-foreign key links. **(d)** Initial node features are extracted from each row in each table using modality-specific neural networks. Then a message passing graph neural network computes relation-aware node embeddings, a model head produces predictions for training table entities, and errors are backpropogated.

2.1. Relational Data

A relational database $(\mathcal{T}, \mathcal{L})$ is comprised of a collection of tables $\mathcal{T} = \{T_1, \ldots, T_n\}$, and links between tables $\mathcal{L} \subseteq \mathcal{T} \times \mathcal{T}$ (*cf.* Figure 2a). A link $L = (T_{\text{fkey}}, T_{\text{pkey}}) \in \mathcal{L}$ between tables exists if a foreign key column in T_{fkey} points to a primary key column of T_{pkey}. Each table is a set $T = \{v_1, ..., v_{n_T}\}$, whose elements $v_i \in T$ are called rows, or entities (*cf.* Figure 2b). Each entity $v \in T$, has four constituent parts $v = (p_v, \mathcal{K}_v, x_v, t_v)$:

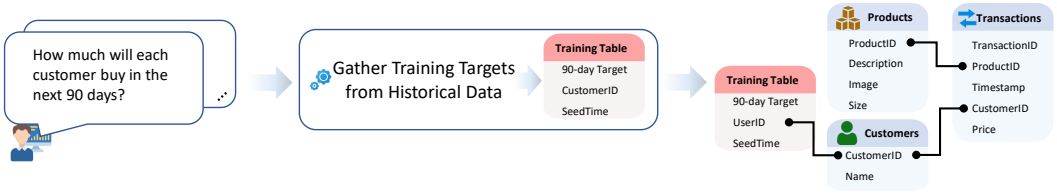
1. **Primary key** p_v, that uniquely identifies the entity v.
2. **Foreign keys** $\mathcal{K}_v \subseteq \{p_{v'} : v' \in T' \text{ and } (T, T') \in \mathcal{L}\}$, defining links between element $v \in T$ to elements $v' \in T'$, where $p_{v'}$ is the primary key of an entity v' in table T'.
3. **Attributes** x_v, holding the information of the entity.
4. **Timestamp** An optional timestamp t_v, indicating the time an event occurred.

For example, the TRANSACTIONS table in Figure 2a has the primary key (TRANSACTIONID), two foreign keys (PRODUCTID and CUSTOMERID), one attribute (PRICE), and timestamp column (TIMESTAMP). Similarly, the PRODUCTS table has the primary key (PRODUCTID), no foreign keys, attributes (DESCRIPTION, IMAGE and SIZE), and no timestamp. The connection between foreign keys and primary keys is given by black connecting lines in Figure 2.

In general, the attributes in table T contain a tuple of d_T values: $x_v = (x_v^1, \ldots, x_v^{d_T})$. Critically, all entities in the same table have the same columns (values may be absent). For example, the PRODUCTS table from Fig. 2a contains three different attributes: the *product description* (text type), the *image* of the product (image type), and the *size* of the product (numerical type). Each of these types has their own encoders as discussed in Sec. 3.4.3.

Fact and Dimension Tables. Tables are categorized into two types, *fact* or *dimension*, with complementary roles (Garcia-Molina et al., 2008). Dimension tables provide contextual information, such as biographical information, macro statistics (such as number of beds in a hospital), or immutable properties, such as the size of a product (as in the PRODUCTS table in Figure 2a). Fact tables record interactions between other entities, such as all patient admissions to a hospital, or all customer transactions (as in the TRANSACTIONS table in Figure 2a). Since entities can interact repeatedly, fact tables often contain the majority of rows in a relational database. Typically, features in dimension tables are static over their whole lifetime, while fact tables usually have timestamped entities.

Temporality as a First-Class Citizen. Relational data evolves over time as events occur and are recorded. This is captured by the (optional) timestamp t_v attached to each en-



(a) Define Tasks **(b)** Training Table Generation **(c)** Link to Relational Tables

Figure 3. **Predictive Task Definition.** A task over relational data is defined by attaching an additional *training table* to the existing linked tables. A training table entity specifies **(a)** ground truth label computed from historical information **(b)** the entity ID(s) the labels correspond to, and **(c)** a timestamp that controls what data the model can use to predict this label.

tity v. For example, each transaction in TRANSACTIONS has a time stamp. Furthermore, many *tasks* of interest involve forecasting future events. For example, how much will a customer spend in next k days. It is therefore essential that time is conferred a special status unlike other attributes. Our formulation, introduced in Section 3, achieves this through a temporal message passing scheme (similar to (Rossi et al., 2020)), that only permits nodes to receive messages from neighbors with earlier timestamps.

2.2. From Task to Training Table

Many practically interesting machine learning tasks defined over relational databases involve predicting the future state of the entities of interest. Given a task we wish to solve, how can we create *ground truth labels* for model training?

Our key insight is that we can generate training labels using historical data. For instance, at time t, ground truth labels for predicting "how much each customer will buy in the next 90 days?" are computed by summing up each customer's spending within the interval t and $t + 90$ days. Importantly, as long as $t + 90$ is less than the most recent timestamp in the database, then these ground truth labels can be computed *purely from historical data* without any need for external annotation. Further, by choosing different time points t across the database time horizon, it is possible to naturally compute many ground truth training labels for each entity.

To hold the labels for a new predictive task, we introduce a new table known as a *training table* T_{train} (Fig. 3). Each entity $v = (\mathcal{K}_v, t_v, y_v)$ in the training table T_{train} has three components: (1) A (set of) foreign keys $\mathcal{K}_v$ indicating the entities the training example is associated to, (2) a timestamp t_v, and (3) the ground truth label itself y_v. In contrast to tabular learning settings, the training table does *not* contain input data x_v. The training table is linked to the main relational database $(\mathcal{T}, \mathcal{L})$ by updating: (1) the tables to $\mathcal{T} \cup \{T_{\text{train}}\}$, and (2) the links between tables to $\mathcal{L} \cup \mathcal{L}_{T_{\text{train}}}$, where $\mathcal{L}_{T_{\text{train}}}$ specifies tables that keys $\mathcal{K}_v$ point to.

As discussed in Sec. 2.1, careful handling of what data the model sees during training is crucial in order to ensure temporal leakage does not happen. This is achieved using the training timestamp. When the model is trained to output target y_v for entity v with timestamp t_v, temporal consistency is ensured by only permitting the model to receive input information from entities u with timestamp $t_u \leq t_v$ (see Sec. 3.3 for details on training sampling).

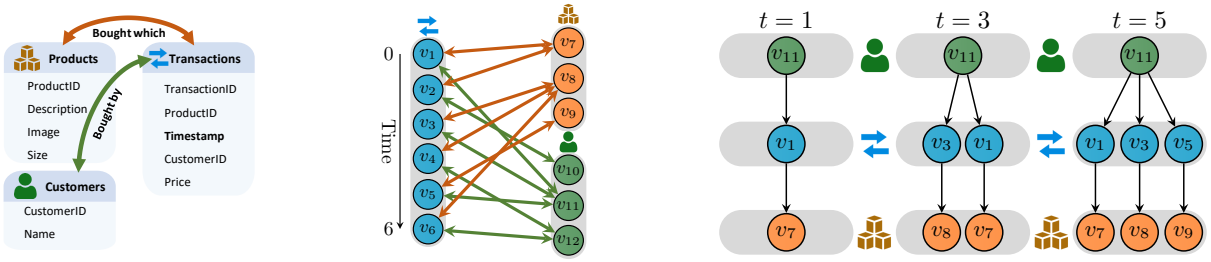
Thus, the purpose of the training table is twofold: to specify training *inputs* and *outputs* of the machine learning model. First, it provides supervision on the model *output* by specifying the the entities and their target training labels. Second, in the case of temporal tasks, the training table specifies the model *input* by specifying the timestamp at which each historical training label is generated. This training table formulation can model a wide range of predictive tasks on relational databases:

- **Node-level prediction tasks** (*e.g.*, multi-class classification, multi-label classification, regression): The training table has three columns (ENTITYID, LABEL, TIME), indicating the foreign key, target label, and timestamp columns.

- **Link prediction tasks**: The training table has columns (SOURCEENTITYID, TARGETENTITYID, LABEL, TIME), indicating the foreign key columns for the source/target nodes, target label, and timestamp.

- **Temporal and static prediction tasks**: Temporal tasks make predictions about the future (and require a seed time), while non-temporal tasks impute missing values (TIME is dropped).

Training Table Generation. In practice, training tables can be computed using time-conditioned SQL queries from historic data in the database. Given a query that describes the prediction targets for all prediction entities, e.g. the sum of sells grouped by products, from time t to time $t + \delta$ in the future, we can move t back in time in fixed intervals to gather historical training, validation and test targets for all entities (*cf.* Fig. 3b). We store t as timestamp for the targets gathered in each step.



(a) Schema Graph (b) Relational Entity Graph (c) Computation Graphs for different time t

Figure 4. **Three different kinds of graphs. (a)** The schema graph arises from the given relational tables. Each node denotes a table, and an edge between tables indicates that primary keys in one are foreign keys in the other. **(b)** The entity graph has one node for each entity in each table, and edges given by primary-foreign key links. The entity graph is heterogeneous with node and edge types defined by the schema graph. The nodes have a timestamp (illustrated by arrow-of-time), originating from the timestamp column of the table. **(c)** Using a temporal sampling strategy and a task description in form of training table containing different time s, we obtain *time-consistent computation graphs* as training examples that naturally respect temporal order and map well to parallel compute.

3. Predictive Tasks as Graph Representation Learning Problems

Here, we formulate a generic graph neural network architecture for solving predictive tasks on relational databases. The following section will first introduce three important graph concepts, which are outlined in Fig. 4: (a) The *schema graph* (*cf.* Sec. 3.1), table-level graph, where one table corresponds to one node. (b) The *relational entity graph* (*cf.* Sec. 3.2), an entity-level graph, with a node for each entity in each table, and edges are defined via foreign-primary key connections between entities. (c) The *time-consistent computation graph* (*cf.* Sec. 3.3), which acts as an explicit training example for graph neural networks. We describe generic procedures to map between graph types, and finally introduce our GNN blueprint for end-to-end learning on relational databases (*cf.* Sec. 3.4).

3.1. Schema Graph

The first graph in our blueprint is the *schema graph* (*cf.* Fig. 4a), which describes the table-level structure of data. Given a relational database $(\mathcal{T}, \mathcal{L})$ as defined in Sec. 2, we let $\mathcal{L}^{-1} = \{(T_{\text{pkey}}, T_{\text{fkey}}) \mid (T_{\text{fkey}}, T_{\text{pkey}}) \in \mathcal{L}\}$ denote its inverse set of links. Then, the schema graph is the graph $(\mathcal{T}, \mathcal{R})$ that arises from the relational database, with node set $\mathcal{T}$ and edge set $\mathcal{R} = \mathcal{L} \cup \mathcal{L}^{-1}$. Inverse links ensure that all tables are reachable within the schema graph. The schema graph nodes serve as type definitions for the heterogeneous relational entity graph, which we define next.

3.2. Relational Entity Graph

To formulate a graph suitable for processing with GNNs, we introduce the *relational entity graph*, which has entity-level nodes and serves as the basis of the proposed framework.

Our relational entity graph is a *heterogeneous graph* $G = (\mathcal{V}, \mathcal{E}, \phi, \psi)$, with node set $\mathcal{V}$ and edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ and type mapping functions $\phi : \mathcal{V} \to \mathcal{T}$ and $\psi : \mathcal{E} \to \mathcal{R}$, where each node $v \in \mathcal{V}$ belongs to a *node type* $\phi(v) \in \mathcal{T}$ and each edge $e \in \mathcal{E}$ belongs to an *edge type* $\psi(e) \in \mathcal{R}$. Specifically, the sets $\mathcal{T}$ and $\mathcal{R}$ from the schema graph define the node and edge types of our relational entity graph.

Given a schema graph $(\mathcal{T}, \mathcal{R})$ with tables $T = \{v_1, ..., v_{n_T}\} \in \mathcal{T}$ as defined in Sec. 2, we define the node set in our relational entity graph as the union of all entries in all tables $\mathcal{V} = \bigcup_{T \in \mathcal{T}} T$. Its edge set is then defined as

$$\mathcal{E} = \{(v_1, v_2) \in \mathcal{V} \times \mathcal{V} \mid p_{v_2} \in \mathcal{K}_{v_1} \text{ or } p_{v_1} \in \mathcal{K}_{v_2}\}, \quad (1)$$

i.e. the entity-level pairs that arise from the primary-foreign key relationships in the database. We equip the relational entity graph with the following key information:

- **Type mapping functions** $\phi : \mathcal{V} \to \mathcal{T}$ and $\psi : \mathcal{E} \to \mathcal{R}$, mapping nodes and edges to respective elements of the schema graph, making the graph *heterogeneous*. We set $\phi(v) = T$ for all $v \in T$ and $\psi(v_1, v_2) = (\phi(v_1), \phi(v_2)) \in \mathcal{R}$ if $(v_1, v_2) \in \mathcal{E}$.
- **Time mapping function** $\tau : \mathcal{V} \to \mathcal{D}$, mapping nodes to its timestamp: $\tau : v \mapsto t_v$ (as defined in Sec. 2.1), introducing time as a central component and establishes the *temporality* of the graph. The value $\tau(v)$ denotes the point in time in which the table row v became available or $-\infty$ in case of non-temporal rows.
- **Embedding vectors** $\mathbf{h}_v \in \mathbb{R}^{d_{\phi(v)}}$ for each $v \in \mathcal{V}$, which contains an embedding vector for each node in the graph. Initial embeddings are obtained via multi-modal column encoders, *cf.* Sec. 3.4.3. Final embeddings are computed via GNNs outlined in Section 3.4.

An example of a relational entity graph for a given schema graph is given in Fig. 4b. The graph contains a node for each row in the database tables. Two nodes are connected if the foreign key entry in one table row links to the primary key entry of another table row. Node and edge types are defined by the schema graph. Nodes resulting from temporal tables carry the timestamp from the respective row, allowing temporal message passing, which is described next.

3.3. Time-Consistent Computational Graphs

Given a relational entity graph and a training table (*cf.* Sec. 2.2), we need to be able to query the graph at specific points in time which then serve as explicit training examples used as input to the model. In particular, we create a subgraph from the relational entity graph induced by the set of foreign keys $\mathcal{K}_v$ and its timestamp t_v of a training example in the training table T_{train}. This subgraph then acts as a local and *time-consistent computation graph* to predict its ground-truth label y_v.

The computational graphs obtained via neighbor sampling (Hamilton et al., 2017) allow the scalability of our proposed approach to modern large-scale relational data with billions of table rows, while ensuring the temporal constraints (Wang et al., 2021). See Appendix B for details.

3.4. Task-Specific Temporal Graph Neural Networks

Given a time-consistent computational graph and its future label to predict, we define a generic multi-stage deep learning architecture as follows:

1. Table-level **column encoders** that encode table row data into initial node embeddings $\mathbf{h}_v^{(0)}$ (*cf.* Sec. 3.4.3).
2. A stack of L **relational-temporal message passing layers** (*cf.* Sec. 3.4.1).
3. A task-specific **model head**, mapping final node embeddings to a prediction (*cf.* Sec. 3.4.2).

The whole architecture, consisting of table-level encoders, message passing layers and task specific model heads can be trained end-to-end to obtain a model for the given task.

3.4.1. Relational-Temporal Message Passing

A message passing operator in the given relational framework needs to respect the heterogeneous nature as well as the temporal properties of the graph. We adopt common *hetereogeneous* message passing (Gilmer et al., 2017; Fey & Lenssen, 2019; Schlichtkrull et al., 2018; Hu et al., 2020) and extend it by a temporal filtering mechanism, as detailed in Appendix A. Given a relational entity graph, initial node embeddings $\{\mathbf{h}_v^{(0)}\}_{v\in\mathcal{V}}$ and an example specific *seed time* $t \in \mathbb{R}$ (*cf.* Sec. 2.2), we obtain a set of node embeddings $\{\mathbf{h}_v^{(L)}\}_{v\in\mathcal{V}}$ by L consecutive applications of message passing, where information flow between nodes can only go forward in time, ensured by a temporal neighbor sampler.

3.4.2. Prediction with Model Heads

The model described so far is task-agnostic and simply propagates information through the relational entity graph to produce node embeddings. We obtain a task-specific model by combining our graph with a training table, leading to specific model heads and loss functions. We distinguish between (but are not limited to) two types of tasks: node-level prediction and link-level prediction.

Node-level Model Head. Given a batch of N node level training table examples $\{(\mathcal{K}, t, y)_i\}_{i=1}^N$ (*cf.* Sec. 2.2), where $\mathcal{K} = \{k\}$ contains the primary key of node $v \in \mathcal{V}$ in the relational entity graph, $t \in \mathbb{R}$ is the seed time, and $y \in \mathbb{R}^d$ is the target value. Then, the node-level model head maps node-level embeddings $\mathbf{h}_v^{(L)}$ to a prediction $\hat{y}$, *i.e.*

$$f : \mathbb{R}^{d_v} \to \mathbb{R}^d, \qquad f : \mathbf{h}_v^{(L)} \mapsto \hat{y}. \tag{2}$$

Link-level Model Head. Similarly, we can define a link-level model head for training examples $\{(\mathcal{K}, t, y)_i\}_{i=1}^N$ with $\mathcal{K} = \{k_1, k_2\}$ containing primary keys of two different nodes $v_1, v_2 \in \mathcal{V}$ in the relational entity graph. A function maps node embeddings $\mathbf{h}_{v_1}^{(L)}$, $\mathbf{h}_{v_2}^{(L)}$ to a prediction, *i.e.*

$$f : \mathbb{R}^{d_{v_1}} \times \mathbb{R}^{d_{v_2}} \to \mathbb{R}^d, \qquad f : (\mathbf{h}_{v_1}^{(L)}, \mathbf{h}_{v_2}^{(L)}) \mapsto \hat{y}. \tag{3}$$

A task-specific loss $L(\hat{y}, y)$ provides gradient signals to all trainable parameters. The presented approach can be generalized to $|\mathcal{K}| > 2$ to specify subgraph-level tasks.

3.4.3. Multi-Modal Node Encoders

The final piece of the pipeline is to obtain the initial entity-level node embeddings $\mathbf{h}_v^{(0)}$ from the multi-modal input attributes $x_v = (x_v^1, \ldots, x_v^{d_T})$. Due to the nature of tabular data, each column element x_v^i lies in its own modality space such as image, text, categorical, and numerical values. Therefore, we use pre-trained modality-specific encoders to embed each attribute into embeddings, and fuse the column-level embeddeings into a single embedding per row.

4. A New Program for Graph Representation Learning

Relational Deep Learning has research opportunities at all levels of the modelling stack, including (pre-)training methods, GNN architectures, multimodality, new graph structues, and scaling to large distributed relational databases. Here we discuss several promising aspects of this research program, aiming to stimulate the graph machine learning community.

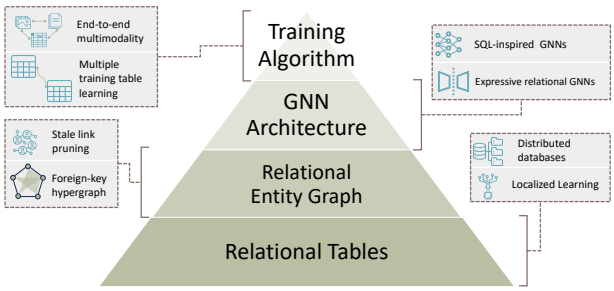


Figure 5. Relational Deep Learning brings new challenges at all levels of the machine learning stack.

4.1. Scaling Relational Deep Learning

Relational databases are often vast, with information distributed across many servers with constrained communication. Accordingly, there is a need for learning algorithms that are compatible with the way data is partitioned across machines. *Horizontal partitioning*, known as sharding, is the most common approach. The most common approach is horizontal partitioning, which splits the overall database into smaller *shards* each stored on different computer nodes. Each shard consists of a unique set of data rows following the same table schema as the original database. This means that similar rows from the same table may lie on different machines, making it difficult to share information.

4.2. Building Graphs from Relational Data

An essential ingredient of Relational Deep Learning is the *relational entity graph* (*cf.* 3.2) which is a graph modelling individual- and link-level information. Whilst a natural choice, we do not propose dogmatically viewing entities as nodes and relations as edges. Instead, the essential property of the relational entity graph is that it is *full-resolution*. That is, each entity and each primary-foreign key link in the relational database corresponds to its own graph-piece, so that the relational database is exactly encoded in graph form. Alternative graph designs can share this property.

Foreign-key Hypergraph. Fact tables often contain entities with a fixed foreign-key pattern (*e.g.*, in Figure 1 a row in a REVIEW table always refers to a CUSTOMER and a PRODUCT foreign key). The relational entity graph views a review as a node, with edges to a customer and product. However, another possibility is to view this as a single hyperedge between review, customer, and product. Alternative graph choices may alter (and improve) information propagation between entities (*cf.* Sec. 4.3).

4.3. GNN Architectures for Relational Data

Viewing a relational database a graphs leads to graphs with structural properties that are consistent across databases. To properly exploit this structure new specialized GNN architectures are needed.

Expressive GNNs for Relational Data. Relational entity graphs (*cf.* Sec. 3.2) obey certain structural constraints. For example, as nodes correspond to entities drawn from one of several tables, the relational entity graph is naturally n-partite, where n is the total number of tables. Unfortunately, recent studies find that many GNN architectures fail to distinguish biconnected graphs (Zhang et al., 2023). Further work is needed to design expressive n-partite graph models. Relational entity graphs also have regularity in edge-connectivity. For instance, in Figure 1, entities in the REVIEW table always refer to one CUSTOMER and one PRODUCT. Consistent edge patterns are described by the structure of the schema graph $(\mathcal{T}, \mathcal{R})$ (*cf.* Sec. 3.1) and may benefit from modified message passing procedures.

Query Language Inspired Models. SQL operations are known to be extremely powerful operations for manipulating relational data. Interestingly, there are close similarities between key SQL queries and the computation process of graph neural networks. For instance, a very common way to combine information across tables T_1, T_2 in SQL is to (1) create a table T_3 by applying a `JOIN` operation to table T_1 and T_2, by matching foreign keys in T_1 to primary keys in T_2, then (2) produce a final table with the same number of rows as T_2 by applying an `AGGREGATE` operation to rows in T_3 with foreign keys pointing to the same entity in T_2. This process *directly* mirrors GNN computations of messages from neighboring nodes, followed by aggregation. This suggests an opportunity for enhancing architectures by designing differentiable computation blocks that algorithmically align (Xu et al., 2020) to other useful SQL operations.

New Message Passing Schemes. New architectures may also improve information propagation between entities. For instance, collaborative filtering methods enhance predictions by identify entities with similar behavior patterns. However, in the relational entity graph, the two related customers may not be directly linked. Instead they are indirectly be linked to one another through links to their respective purchases, which are linked to a particular shared product ID. This means that a standard message passing GNN will require four message passing steps to propagate the information that customer v_1 purchased the same product as customer v_2. New message passing schemes that propagate key information through the model in fewer iterations may be desirable. As well as new message passing schemes, there is also opportunity for new message aggregation methods, such as time-aware aggregation (Yang et al., 2022).

4.4. Training Techniques for Relational Data

By its nature, relational data contains highly overlapping predictive signals and tasks. This interconnectedness is an opportunity for new neural network training methods.

Multi-Task Learning. Many predictive tasks on relational data are distinct but related. For example, predicting customer lifetime value, and forecasting individual product sales both involve anticipating future purchase patterns. In RELBENCH, this corresponds to defining multiple training tables, one for each task, and training a single model jointly on all tasks in order to benefit from shared predictive signals. How to group training tables to leverage their overlap is a promising area for further study.

Multi-Modal Learning. Entities often have attributes covering multiple modalities (*e.g.,* products come with images, descriptions, as well as different categorical and numerical features). The Relational Deep Learning blueprint first extracting entity-level features, which are used as initial node-features for the GNN model. A natural initial entity-level feature extraction approach is to use state-of-the-art pre-trained models This maximizes convenience for graph-focused research, but is likely suboptimal because the entity-level feature extraction model is frozen. This is especially relevant in contexts with unusual data—e.g., specialized medical documentation—that generic pre-trained models will likely fail to extract important details.

Foundation Models. In practice, new predictive tasks on relational data are often specified on-the-fly and require fast responses, precluding costly model training from scratch. To pre-train models, self-supervised labels can be mined from historical data, just as with training table construction. How to specify suitable self-supervised tasks is an important open question. Furthermore, in order to generalize to entirely new relational databases, it is also necessary to design *inductive* models that can deal with new columns types.

5. Related Work

Statistical Relational Learning. Since the foundation of the field of AI, researchers sought to design systems capable of reasoning about entities and their relations, often by explicitly building graph structures (Minsky, 1974). Each new era of AI research also brought its own form of relational learning. A prominent instance is statistical relational learning (De Raedt, 2008), a common form of which seeks to describe objects and relations in terms of first-order logic, fused with graphical models to model uncertainty (Getoor et al., 2001). These descriptions can then be used to generate new "knowledge" through inductive logic programming (Lavrac & Dzeroski, 1994). Markov logic networks, a prominent statistical relational approach, are defined by a collection of first-order logic formula with accompanying scalar weights (Richardson & Domingos, 2006). This information is then used to define a probability distribution over possible worlds (via Markov random fields) which enables probabilistic reasoning about the truth of new formulae. We see Relational Deep Learning as inheriting this lineage.

Tabular Machine Learning. Tree based methods, notably XGBoost (Chen & Guestrin, 2016), remain key workhorses of enterprise machine learning systems due to their scalability and reliability. In parallel, efforts to design deep learning architectures for tabular data have continued (Huang et al., 2020; Arik & Pfister, 2021; Gorishniy et al., 2021; 2022; Chen et al., 2023), but have struggled to clearly dominate tree-based methods (Shwartz-Ziv & Armon, 2022). The vast majority of tabular machine learning focuses on the single tables, which we argue forgoes use of the rich interconnections between relational data. As such, it does not address the key problem for relational data, which is how to get the data from a multi-table to a single table representation.

Deep Learning on Relational Data. Proposals to use message passing neural networks on relational data have occasionally surfaced within the research community. In particular, (Schlichtkrull et al., 2018; Cvitkovic, 2019; Šír, 2021), and (Zahradník et al., 2023) make the connection between relational data and graph neural networks and explore it with different network architectures. Here we focus on the components needed to establish this new area and attract broader interest: (1) a clearly scoped design space for neural network architectures on relational data, (2) a carefully chosen suite of benchmark databases and predictive tasks around which the community can center its efforts, (3) standardized data loading and splitting, so that temporal leakage does not contaminate experimental results, (4) recognizing time as a first-class citizen, integrated into all sections of the experimental pipeline, including temporal data splitting, time-based forecasting tasks, and temporal-based message passing, and (5) standardized evaluation protocols to ensure comparability between reported results.

6. RELBENCH: A Benchmark for Relational Deep Learning

To validate the effectiveness of relational deep learning, we build RELBENCH. RELBENCH includes: (1) a general data loading library intended to make it easy to load relational databases ready for model training; (2) two initial databases: Amazon review records, and Stack Exchange, and two predictive tasks for each database. Additionally, RELBENCH comes with a GNN implementation for solving these tasks. Due to space limitations, we outline the main features of RELBENCH in Appendix C and preliminary GNN results in Appendix D. Our tests find that RDL produces significant improvements over single-table methods such as XGBoost.

Impact Statement

Relational deep learning broadens the applicability of graph machine learning to include relational databases. Whilst the blueprint is general, and can be applied to a wide variety of tasks, including potentially hazardous ones, we have taken steps to focus attention of potential positive use cases. Specifically, the beta version of RELBENCH considers two databases, Amazon products, and Stack Exchange, that are designed to highlight the usefulness of RDL for driving online commerce and online social networks. Future releases of RELBENCH will continue to expand the range of databases into domains we reasonably expect to be positive, such as biomedical data and sports fixtures. We hope these concrete steps ensure the adoption of RDL for purposes broadly beneficial to society.

References

Arik, S. Ö. and Pfister, T. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 6679–6687, 2021.

Chamberlin, D. D. and Boyce, R. F. Sequel: A structured english query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, pp. 249–264, 1974.

Chen, K.-Y., Chiang, P.-H., Chou, H.-R., Chen, T.-W., and Chang, T.-H. Trompt: Towards a better deep neural network for tabular data. In *International Conference on Machine Learning (ICML)*, 2023.

Chen, T. and Guestrin, C. Xgboost: A scalable tree boosting system. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 785–794, 2016.

Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

Cvitkovic, M. Supervised learning on relational databases with graph neural networks. *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

De Raedt, L. *Logical and relational learning*. Springer Science & Business Media, 2008.

Fey, M. and Lenssen, J. E. Fast graph representation learning with pytorch geometric. *ICLR 2019 (RLGM Workshop)*, 2019.

Garcia-Molina, H., Ullman, J. D., and Widom, J. *Database Systems: The Complete Book*. Prentice Hall Press, USA, 2 edition, 2008. ISBN 9780131873254.

Getoor, L., Friedman, N., Koller, D., and Pfeffer, A. Learning probabilistic relational models. *Relational data mining*, pp. 307–335, 2001.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning (ICML)*, pp. 1263–1272, 2017.

Gorishniy, Y., Rubachev, I., Khrulkov, V., and Babenko, A. Revisiting deep learning models for tabular data. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 34, pp. 18932–18943, 2021.

Gorishniy, Y., Rubachev, I., and Babenko, A. On embeddings for numerical features in tabular deep learning. *Advances in Neural Information Processing Systems*, 35: 24991–25004, 2022.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Hu, W., Yuan, Y., Zhang, Z., Nitta, A., Cao, K., Kocijan, V., Leskovec, J., and Fey, M. Pytorch frame: A modular framework for multi-modal tabular learning. *arXiv preprint arXiv:2404.00776*, 2024.

Hu, Z., Dong, Y., Wang, K., and Sun, Y. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, pp. 2704–2710, 2020.

Huang, X., Khetan, A., Cvitkovic, M., and Karnin, Z. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

Johnson, A. E., Pollard, T. J., Shen, L., Lehman, L.-w. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., and Mark, R. G. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

Kaggle. Kaggle Data Science & Machine Learning Survey, 2022. Available: https://www.kaggle.com/code/paultimothymooney/kaggle-survey-2022-all-results/notebook.

Kapoor, S. and Narayanan, A. Leakage and the reproducibility crisis in machine-learning-based science. *Patterns*, 4 (9), 2023.

Lavrac, N. and Dzeroski, S. Inductive logic programming. In *WLP*, pp. 146–160. Springer, 1994.

Minsky, M. A framework for representing knowledge, 1974.

PubMed. National Center for Biotechnology Information, U.S. National Library of Medicine, 1996. Available: https://www.ncbi.nlm.nih.gov/pubmed/.

Richardson, M. and Domingos, P. Markov logic networks. *Machine learning*, 62:107–136, 2006.

Rossi, E., Chamberlain, B., Frasca, F., Eynard, D., Monti, F., and Bronstein, M. Temporal graph networks for deep learning on dynamic graphs. *ICML Workshop on Graph Representation Learning 2020*, 2020.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015.

Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., and Welling, M. Modeling relational data with graph convolutional networks. In Gangemi, A., Navigli, R., Vidal, M.-E., Hitzler, P., Troncy, R., Hollink, L., Tordai, A., and Alam, M. (eds.), *The Semantic Web*, pp. 593–607, Cham, 2018. Springer International Publishing.

Shwartz-Ziv, R. and Armon, A. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

Šír, G. *Deep Learning with Relational Logic Representations*. Czech Technical University, 2021.

Varma, M. and Zisserman, A. A statistical approach to texture classification from single images. *International journal of computer vision*, 62:61–81, 2005.

Wang, Y., Cai, Y., Liang, Y., Ding, H., Wang, C., and Hooi, B. Time-aware neighbor sampling for temporal graph networks. In *arXiv pre-print*, 2021.

Xu, K., Li, J., Zhang, M., Du, S. S., Kawarabayashi, K.-i., and Jegelka, S. What can neural networks reason about? In *International Conference on Learning Representations (ICLR)*, 2020.

Yang, Z., Ding, M., Xu, B., Yang, H., and Tang, J. Stam: A spatiotemporal aggregation method for graph neural network-based recommendation. In *Proceedings of the ACM Web Conference 2022*, pp. 3217–3228, 2022.

Zahradník, L., Neumann, J., and Šír, G. A deep learning blueprint for relational databases. In *NeurIPS 2023 Second Table Representation Learning Workshop*, 2023.

Zhang, B., Luo, S., Wang, L., and He, D. Rethinking the expressive power of gnns via graph biconnectivity. In *International Conference on Learning Representations (ICLR)*, 2023.

A. Temporal, Heterogeneous Message Passing

Message-Passing Graph Neural Networks (MP-GNNs) (Gilmer et al., 2017; Fey & Lenssen, 2019) are a generic computational framework to define deep learning architectures on graph-structered data. Given a heterogeneous graph $G = (\mathcal{V}, \mathcal{E}, \phi, \psi)$ with initial node embeddings $\{\mathbf{h}_v^{(0)}\}_{v \in \mathcal{V}}$, a single message passing iteration computes updated features $\{\mathbf{h}_v^{(i+1)}\}_{v \in \mathcal{V}}$ from features $\{\mathbf{h}_v^{(i)}\}_{v \in \mathcal{V}}$ given by the previous iteration. One iteration takes the form:

$$\mathbf{h}_v^{(i+1)} = f(\mathbf{h}_v^{(i)}, \{\!\{ g(\mathbf{h}_w^{(i)}) \mid w \in \mathcal{N}(v) \}\!\}), \tag{4}$$

where f and g are arbitrary differentiable functions with optimizable parameters and $\{\!\{\cdot\}\!\}$ an permutation invariant set aggregator, such as mean, max, sum, or a combination. Heterogeneous message passing (Schlichtkrull et al., 2018; Hu et al., 2020) is a *nested* version of Eq. 4, adding an aggregation over all incoming edge types to learn distinct message types:

$$\mathbf{h}_v^{(i+1)} = f_{\phi(v)}\Big(\mathbf{h}_v^{(i)}, \Big\{\!\!\Big\{ f_R(\{\!\{ g_R(\mathbf{h}_w^{(i)}) \mid w \in \mathcal{N}_R(v) \}\!\}) \,\Big|\, \forall R = (T, \phi(v)) \in \mathcal{R} \Big\}\!\!\Big\}\Big), \tag{5}$$

where $\mathcal{N}_R(v) = \{w \in \mathcal{V} \mid (w, v) \in \mathcal{E} \text{ and } \psi(w, v) = R\}$ denotes the ***R*-specific *neighborhood*** of node $v \in \mathcal{V}$. This formulation supports a wide range of different graph neural network operators, which define the specific form of functions $f_{\phi(v)}$, f_R, g_R and $\{\!\{\cdot\}\!\}$ (Fey & Lenssen, 2019).

Temporal Message Passing. Given a relational entity graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{T}, \mathcal{R})$ with attached mapping functions ψ, ϕ, τ and initial node embeddings $\{\mathbf{h}_v^{(0)}\}_{v \in \mathcal{V}}$ and an example specific *seed time* $t \in \mathbb{R}$ (*cf.* Sec. 2.2) , we obtain a set of deep node embeddings $\{\mathbf{h}_v^{(L)}\}_{v \in \mathcal{V}}$ by L consecutive applications of Eq. 5, where we additionally filter R-specific neighborhoods based on their timestamp, *i.e.* replace $\mathcal{N}_R(v)$ with

$$\mathcal{N}_R^{\leq t}(v) = \{w \in \mathcal{V} \mid (w, v) \in \mathcal{E}, \psi(w, v) = R, \text{and } \tau(w) \leq t\},$$

realized by the temporal sampling procedure presented in Sec. 3.3. The formulation naturally respects time by only aggregating messages from nodes that were available before the given seed time s. The given formulation is agnostic to specific implementations of message passing and supports a wide range of different operators.

B. Time-Consistent Computation Graphs

Given a number of hops L to sample, a seed node $v \in \mathcal{V}$, and a timestamp t induced by a training example, the computation graph is defined as $G_{\text{comp}} = (\mathcal{V}_{\text{comp}}, \mathcal{E}_{\text{comp}})$ as the output of Alg. 1. The algorithm traverses the graph starting from the seed node v for L iterations. In iteration i, it gathers a maximum of m_i neighbors available up to timestamp t, using one of three selection strategies:

- **Uniform temporal sampling** selects uniformly sampled random neighbors.
- **Ordered temporal sampling** takes the latest neighbors, ordered by time τ.
- **Biased temporal sampling** selects random neighbors sampled from a multinomial probability distribution induced by τ. For instance, sampling can be performed proportional to relative neighbor time or biased towards specific important historical moments.

The temporal neighbor sampling is performed purely on the graph structure of the relational entity graph, without requiring initial embeddings $\mathbf{h}_v^{(0)}$. The bounded size of computation graph G_{comp} allows for efficient mini-batching on GPUs, independent of relational entity graph size. In practice, we perform temporal neighbor sampling on-the-fly, which allows us to operate on a shared relational entity graph across all training examples, from which we can then restore local and historical snapshots very efficiently. Examples of computation graphs are shown in Fig. 4c.

We introduce RELBENCH, an open benchmark for Relational Deep Learning. The goal of RELBENCH is to facilitate scalable, robust, and reproducible machine learning research on relational tables. RELBENCH curates a diverse set of large-scale, challenging, and realistic benchmark databases and defines meaningful predictive tasks over these databases. In addition, RELBENCH develops a Python library for loading relational tables and tasks, constructing data graphs, and providing unified evaluation for predictive tasks. It also integrates seamlessly with existing Pytorch Geometric and PyTorch Frame functionalities. In its beta release[1], we announce the first two real-world relational databases, each with two curated

[1]Website: https://relbench.stanford.edu/

Algorithm 1 Time-Consistent Computation Graph

Input: Relational entity graph $G = (\mathcal{V}, \mathcal{E})$, number of hops L, seed node $v_0 \in \mathcal{V}$, seed time $t \in \mathbb{R}$
Input: Neighborhood sizes $(m_1, ..., m_L) \in \mathbb{N}^L$
Output: Computation graph $G_{\text{comp}} = (\mathcal{V}_{\text{comp}}, \mathcal{E}_{\text{comp}})$
  $\mathcal{V}_0 \leftarrow \{v_0\}, \quad \mathcal{E}_0 \leftarrow \emptyset$
  for $i \in \{1, ..., L\}$ **do**
    for $v \in \mathcal{V}_{i-1}$ **do**
      $\mathcal{E}_i \leftarrow \text{SELECT}_{m_i}(\{(w, v) \in \mathcal{E} \mid \tau(v) \leq t\})$ ▷ Select a maximum of m_i filtered edges
      $\mathcal{V}_i \leftarrow \{w \in \mathcal{V} \mid (w, v) \in \mathcal{E}_i\}$ ▷ Gather nodes for the sampled edges
    end for
  end for
  $\mathcal{V}_{\text{comp}} \leftarrow \bigcup_{i=1}^{L} \mathcal{V}_i, \quad \mathcal{E}_{\text{comp}} \leftarrow \bigcup_{i=1}^{L} \mathcal{E}_i$

predictive tasks.

In the subsequent sections (Sec. C.2 and C.3), we describe in detail the two relational databases and the predictive tasks. For each database, we show its entity relational diagrams and important statistics. For each task, we define the task formulation, entity filtering, significance of the task, and also unified evaluation metric. Finally, we demonstrate the usage of the RELBENCH's package in Sec. C.

C. RELBENCH Package

The RELBENCH package is designed to allow easy and standardized access to Relational Deep Learning for researchers to push the state-of-the-art of this emerging field. It provides Python APIs to (1) download and process relational databases and their predictive tasks; (2) load standardized data splits and generate relevant train/validation/test tables; (3) evaluate on machine learning predictions. It also provides a flexible ecosystems of supporting tools such as automatic conversion to PyTorch Geometric graphs and integration with Pytorch Frame to produce embeddings for diverse column types. We additionally provide end-to-end scripts for training using RELBENCH package with GNNs and XGBoost (Chen & Guestrin, 2016). We temporarily withhold the website link to RELBENCH to preserve anonymity.

C.1. Temporal Splitting

Every dataset in RELBENCH has a validation timestamp t_{val} and a test timestamp t_{test}. These are shared for all tasks in the dataset. The test table for any task comprises of labels computed for the time window from t_{test} to $t_{\text{test}} + \delta$, where the window size δ is specified for each task. Thus the model must make predictions using only information available up to time t_{test}. Accordingly, to prevent accidental temporal leakage at test time RELBENCH only provides database rows with timestamps up to t_{test} for training and validation purposes. RELBENCH also provides default train and validation tables. The default validation table is constructed similar to the test table, but with the time window being t_{val} to $t_{\text{val}} + \delta$. To construct the default training table, we first sample time stamps t_i starting from $t_{\text{val}} - \delta$ and moving backwards with a stride of δ. This allows us to benefit from the latest available training information. Then for each t_i, we apply an entity filter to select the entities of interest (e.g., active users). Finally for each pair of timestamp and entity, we compute the training label based on the task definition. Users can explore other ways of constructing the training or validation table, for example by sampling timestamps with shorter strides to get more labels, as long as information after t_{val} is not used for training.

C.2. `rel-amazon`: Amazon product review e-commerce database

Database overview. The `rel-amazon` relational database stores product and user purchasing behavior across Amazon's e-commerce platform. Notably, it contains rich information about each product and transaction. The product table includes price and category information; the review table includes overall rating, whether the user has actually bought the product, and the text of the review itself. We use the subset of book-related products. The entity relationships are described in Fig. 6.

Dataset statistics. `rel-amazon` covers 3 relational tables and contains 1.85M customers, 21.9M reviews, 506K products. This relational database spans from 1996-06-25 to 2018-09-28. The validation timestamp t_{val} is set to 2014-01-21 and the testing timestamp t_{test} is 2016-01-01. Thus, tasks can have a window size up to 2 years.

Figure 6. `rel-amazon` contains two dimension tables (customers and products) and one fact table (reviews). Each review has a customer and a product foreign key.

C.2.1. REL-AMAZON-LTV: PREDICT THE LIFE TIME VALUE (LTV) OF A USER

Task definition: Predict the life time value of a user, defined as the total number of products that the user will buy and review in the next 2 years.

Entity filtering: We filter on active users defined as users that wrote review in the past two years before the timestamp.

Task significance: By accurately forecasting LTV, the e-commerce platform can gain insights into user purchasing patterns and preferences, which is essential when making strategic decisions related to marketing, product recommendations, and inventory management. Understanding a user's future purchasing behavior helps in tailoring personalized shopping experiences and optimizing product assortments, ultimately enhancing customer satisfaction and loyalty.

Machine learning task: Regression. The target ranges from 0-7922in the given time window in the training table.

Evaluation metric: Mean Absolute Error (MAE).

C.2.2. REL-AMAZON-CHURN: PREDICT IF THE USER CHURNS

Task definition: Predict if the user will not buy any product in the next 2 years.

Entity filtering: We filter on active users defined as users that wrote review in the past two years before the timestamp.

Task significance: Predicting churn accurately allows companies to identify potential risks of customer attrition early on. By understanding which customers are at risk of disengagement, businesses can implement targeted interventions to improve customer retention. This may include personalized marketing, tailored offers, or enhanced customer service. Effective churn prediction enables businesses to maintain a stable customer base, ensuring sustained revenue streams and facilitating long-term planning and resource allocation.

Machine learning task: Binary classification. The label is 1 when user churns and 0 vice versus.

Evaluation metric: Average precision (AP).

C.3. `rel-stackex`: Stack exchange question-and-answer website database

Database overview. Stack Exchange is a network of question-and-answer websites on topics in diverse fields, each site covering a specific topic, where questions, answers, and users are subject to a reputation award process. The reputation system allows the sites to be self-moderating. In our benchmark, we use the stats-exchange site. We derive from the raw data dump from 2023-09-12. Figure 7 shows its entity relational diagrams.

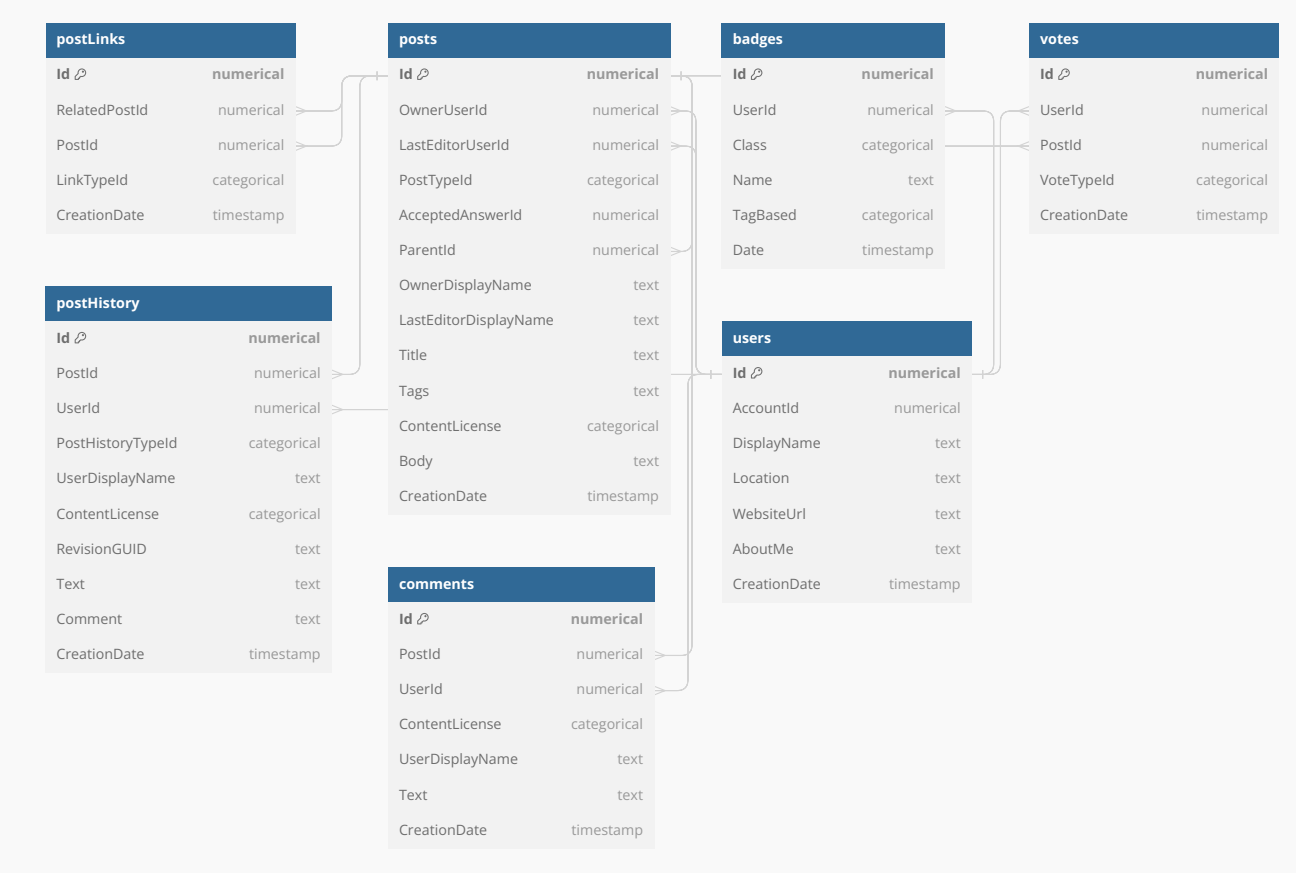


Figure 7. Entity relational diagrams of `Stack-Exchange`.

Dataset statistics. `rel-stackex` covers 7 relational tables and contains 333K users, 415K posts, 794K comments, 1.67M votes, 103K post links, 590K badges records, 1.49M post history records. This relational database spans from 2009-02-02 to 2023-09-03. The validation timestamp t_{val} is set to be 2019-01-01 and the testing timestamp t_{test} is set to be 2021-01-01. Thus, the maximum time window size for predictive task is 2 years.

C.3.1. REL-STACKEX-ENGAGE: PREDICT IF A USER WILL BE AN ACTIVE CONTRIBUTOR TO THE SITE

Task definition: Predict if the user will make any contribution, defined as vote, comment, or post, to the site in the next 2 years.

Entity filtering: We filter on active users defined as users that have made at least one comment/post/vote before the timestamp.

Task significance: By accurately forecasting the levels of user contribution, website administrators can effectively gauge and oversee user activity. This insight allows for well-informed choices across various business aspects. For instance, it aids in preempting and mitigating user attrition, as well as in enhancing strategies to foster increased user interaction and involvement. This predictive task serves as a crucial tool in optimizing user experience and sustaining a dynamic and engaged user base.

Machine learning task: Binary classification. The label is 1 when user contributes to the site and 0 otherwise.

Evaluation metric: Average Precision (AP).

C.3.2. REL-STACKEX-VOTES: PREDICT THE NUMBER OF UPVOTES A QUESTION WILL RECEIVE

Task definition: Predict the popularity of a question post in the next six months. The popularity is defined as the number of upvotes the post will receive.

Entity filtering: We filter on question posts that are posted recently in the past 2 years before the timestamp. This ensures that we do not predict on old questions that have been outdated.

Task significance: Predicting the popularity of a question post is valuable as it empowers site managers to predict and prepare for the influx of traffic directed towards that particular post. This foresight is instrumental in making strategic business decisions, such as curating question recommendations and optimizing content visibility. Understanding which posts are likely to attract more attention helps in tailoring the user experience and managing resources effectively, ensuring that the most engaging and relevant content is highlighted to maintain and enhance user engagement.

Machine learning task: Binary classification. The label is 1 when a post recieves one or more upvote, and 0 otherwise.

Evaluation metric: Average Precision (AP).

D. Preliminary Results

This section reports preliminary results on the four predictive tasks outlined in Appendix C. The aim is to provide a preliminary proof of concept that our proposed approach can significantly outperform single-table approaches such as XGBoost.

Baselines. We compare to XGBoost on a single table (i.e., no feature engineering) and to several naive input-independent baselines. For binary classification tasks we compare to: *random*: for each entity simply sample a value in $[0, 1]$ to be the estimated probability of class 1; *majority*: always predict 1 if over 50% of training targets are 1, and 0 otherwise. For regression tasks we compare to regression-suitable naive baselines: *global zero*: always predict zero; *global mean*: always predict the mean of the training targets; *global median*: always predict the median of training targets; *entity mean*: predict the mean over all training targets associated to the entity in question (there are multiple per-entity due to multiple timestamps being used to generate training data); *entity median*: predict the median over all training targets associated to the entity in question

Results. Across all tasks we see that the GNN-based method significantly outperforms XGBoost and naive baselines. We note that two cases, test accuracy is higher for a naive majority vote. This happens when the label distribution is highly unbalanced, so a majority vote can get good performance by virtue of the unbalancedness. In such cases it is more appropriate to use AP and ROCAUC metrics to compare models. We include test accuracy nonetheless as an interesting additional statistic. We emphasize that our XGBoost baseline is on a single table—the entity in question. We do not do any SQL-based feature engineering. Accordingly the significant difference between the GNN model and XGBoost show the performance lift in the non-manually feature engineered setting.

Table 1. `rel-stackex-engage`. XGBoost performed over the user table joined with training table.

Method	Test ROCAUC	Test AP	Test Accuracy
Random	0.4988	0.0589	0.5011
Majority	0.5000	0.0591	0.9408
XGBoost	0.6143	0.1346	0.9380
XGBoost with AR labels	0.8200	0.3215	0.9195
GNN	**0.8754**	**0.4513**	**0.9479**

Table 2. `rel-stackex-votes`. XGBoost performed over the posts table joined with training table.

Method	Test ROCAUC	Test AP	Test Accuracy
Random	0.4956	0.1347	0.4999
Majority	0.5000	0.1353	**0.8646**
XGBoost	0.6213	0.1989	0.8431
XGBoost with AR labels	0.8200	0.3215	0.9195
GNN	**0.7086**	**0.2853**	0.8489

Table 3. `rel-amazon-churn`. XGBoost performed over the user table joined with training table.

Method	Test ROCAUC	Test AP	Test Accuracy
Random	0.4990	0.3199	0.4999
Majority	0.5000	0.3197	**0.6802**
XGBoost	0.512	0.3295	0.674
GNN	**0.6556**	0.**4536**	0.6655

Table 4. `rel-amazon-ltv`. XGBoost performed over the user table joined with training table.

Method	Test MAE	Test RMSE
Global Zero	3.8486	13.3635
Global Mean	4.2549	12.7994
Global Median	3.4881	13.1105
Entity Mean	4.2174	11.9572
Entity Median	4.2162	12.0748
XGBoost	3.4881	13.1105
GNN	**3.12371**	**10.6085**