Relational Deep Learning: Graph Representation Learning on Relational Databases

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

[*]Equal contribution. Listed in alphabetical order.

[1]Stanford University
[2]Kumo.AI
[3]Max Planck Institute for Informatics
[4]Yale University
[5]University of Illinois at Urbana-Champaign

RELBENCH: https://relbench.stanford.edu

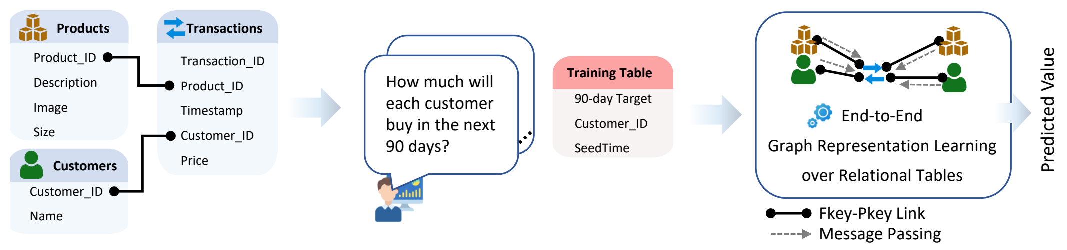
Abstract

Much of the world's most valued data is stored in data warehouses, where the data is spread across many tables connected by primary-foreign key relations. However, building machine learning models using this data is both challenging and time consuming. The core problem is that no machine learning method is capable of learning directly on the data spread across multiple relational tables. Current methods can only learn from a single table, so the data must first be joined and aggregated into a single training table, the process known as feature engineering. Here we introduce an end-to-end deep representation learning approach to directly learn on data spread across multiple tables. We name our approach *Relational Deep Learning*. The core idea is to view relational tables as a heterogeneous graph, with a node for each row in each table, and edges specified by primary-foreign key relations. Message Passing Neural Networks can then automatically learn across multiple tables to extract representations that leverage all input data, without any manual feature engineering. To facilitate research, we also develop RELBENCH, a set of benchmark datasets and an implementation of Relational Deep Learning. The data covers a wide spectrum, from discussions on Stack Exchange to book reviews on the Amazon Product Catalog. Overall, we define a new research area that generalizes graph machine learning and broadens its applicability to a wide set of AI use cases.

1 Introduction

The information age is driven by data stored in ever-growing databases and data warehouses that have come to underpin nearly all technology stacks. Data warehouses typically store information in multiple tables of data, with entities connected using primary-foreign key relations, and managed using powerful query languages such as SQL [Codd, 1970, Chamberlin and Boyce, 1974]. For this reason, data warehouses underpin many of today's large information systems, including e-commerce, social media, banking systems, healthcare, manufacturing, and open-source scientific knowledge repositories [Johnson et al., 2016, PubMed, 1996].

Many predictive problems over relational data have significant implications for human decision making. A hospital wants to predict the risk of discharging a patient; an e-commerce company



(a) Relational Tables **(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 database of relational tables 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 GNN trained on the training table. The task can be node level (as in this illustration), or link-level or higher-order.

wishes to forecast future sales of each of their products; a telecommunications provider wants to predict which customers will continue using their services (churn); and a music streaming platform must decide which songs to recommend to a user. Behind each of these problems there is a rich relational schema of different relational tables, and many machine learning models are built using this data [Kaggle, 2022].

However, existing learning paradigms, notably tabular learning, cannot be directly applied to interlinked relational tables. Instead, a manual feature engineering step is first taken, where a data scientist uses domain knowledge to manually join and aggregate tables to generate features in a regular single table format. To illustrate this, consider a simple e-commerce schema (Fig. 1) of three tables: CUSTOMERS, TRANSACTIONS and PRODUCTS, where CUSTOMERS and PRODUCTS tables link into the TRANSACTIONS table via primary-foreign keys, and the task is to predict if a customer is going to churn (*i.e.*, make zero transactions in the next 30 days). In this case, the 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", "sum of purchase amounts of a given customer in the last 30 days", "number of purchases on a Sunday", "sum of purchase amounts on a Sunday", "number of purchases on a Monday", and so on. The computed customer features are then stored in a single table, ready for tabular machine learning. Another issue is the *temporal* nature of the churn predictive tasks. As new transactions appear, this means that the customer's churn label may change from day to day and also the customer's features may change, so they need to be recomputed for each day. Overall, the issue of temporality adds computational cost and further complexity, which often results in bugs, information leakage and the so-called "time travel".

In other words, predictive tasks on relational data are solved using a combination of hand-crafted features and learned models. There are several issues with feature engineering: (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) by forcing data into a single table, information is aggregated into lower-granularity features, thus losing out on valuable fine-grain signal; (5) whenever data distribution changes or drifts, current features become obsolete 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 spatially aware features, followed by non-spatially aware models such as SVMs or nearest neighbor search [Varma and Zisserman, 2005]. The deep learning revolution has had a huge impact in many fields, including computer vision, natural language processing, and speech, and has led to super-human performance in many tasks. In all cases, the key was to move from manual feature engineering and handcrafted systems to full-neural data-driven end-to-end representation learning systems. For relational data, this transition has not yet occurred, as existing tabular deep learning approaches still heavily rely on manual feature engineering. Consequently, there remains a huge amount of untapped predictive signal.

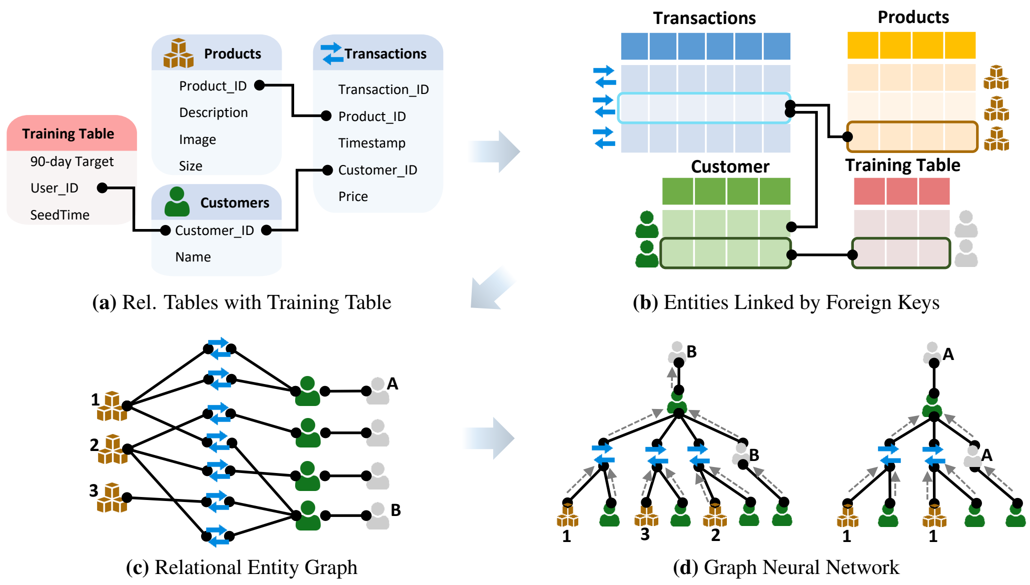


Figure 2: **Relational Deep Learning Pipeline.** **(a)** Relational tables are given and a task defined with a training table. **(b)** Relational tables contain individual entities that are linked by foreign-primary key relations. **(c)** Relational data can be viewed as a single *Relational Entity graph*, 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.

Here we introduce *Relational Deep Learning*, a blueprint for fulfilling the need for an end-to-end deep learning paradigm for relational tables (Fig. 1). Through end-to-end representation learning we fully utilize the rich predictive signals available in relational tables. The core of our approach is to represent relational tables as a heterogeneous *Relational Entity Graph*, where each row defines a node, columns define node features, and primary-foreign key relations define edges. Graph Neural Networks (GNNs) can then be applied to build end-to-end predictive models.

Predictive tasks are specified on relational data by introducing a *training table* that holds label information (Fig. 1b). Training tables have two critically important characteristics. First, labels can be automatically computed from historical relational data, without any need for outside annotation; second, they may contain any number of foreign keys, permitting many task types including entity level (1 key, as in Fig. 1b), link-level tasks such as recommendation (2 keys) and multi-entity tasks (>2 keys). Training tables permit many different types of prediction targets, including multi-class, multi-label, regression and more, ensuring high task generality.

In all, our model pipeline has four main steps (Fig. 2): (1) a predictive machine learning task is specified by adding a *training table* to the database containing training labels computed from historic data in the relational database, (2) entity-level features are extracted from each row in each table to serve as initial node features, (3) new 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 for training data, and errors are backpropogated through the network.

Crucially, our models natively integrate temporality by only allowing entities to receive messages from other entities with earlier timestamps. This ensures that learned features are automatically updated during GNN forward pass when new data is collected, and prevents information leakage and time travel bugs. Furthermore, this also stabilizes the generalization across time since models are trained to make predictions at multiple time snapshots by dynamically passing messages between entities at different time snapshots, whilst remaining grounded in a single relational database.

RelBench. To facilitate research into Relational Deep Learning, we introduce RelBench, a benchmarking and evaluation Python package. Data in RelBench covers a broad range of commercial activity, human organization, and natural phenomena. RelBench has the following key modules 1) **Data:** data loading, specifying a predictive task, and (temporal) data splitting, 2) **Model:** transforming data to a graph, building graph neural network predictive models, 3) **Evaluation:** standardized evaluation protocol given a file of predictions. Importantly, data and evaluation modules are deep learning framework agnostic, enabling broad compatibility. Our model module uses PyTorch Geometric [Fey and Lenssen, 2019] to benchmark several popular graph neural network architectures.

For initial beta release, RelBench contains two databases, each with two predictive tasks. The first database is from Stack Exchange, the question-and-answer website, and includes 7 tables such as posts, users, and votes. The tasks are (1) to predict if a user will make a new contribution (post, answer etc.), and (2) to predict the popularity of a new question. The second database is a subset of the Amazon Product Catalog focusing on books. There are three tables: users, products, and reviews. The tasks are (1) to predict the lifetime value of a user, and (2) whether a user will stop using the site.

Our objective is to establish deep learning on relational data as a new subfield of machine learning. We hope that this will be a fruitful research direction, with many opportunities for impactful ideas that make much better use of the rich predictive signal in relational data. This paper lays the ground for future work by making the following main sections:

- **Blueprint.** *Relational Deep Learning*, an end-to-end learnable approach that ultilizes the predictive signals available in relational data, and supports temporal predictions.
- **Benchmarking Package** RelBench, an open-source Python package for benchmarking and evaluating GNNs on relational data. RelBench beta release introduces two relational databases, and specifies two prediction tasks for each.
- **Research Opportunities.** Outlining a new research program for Relational Deep Learning, including multi-task learning, new GNN architectures, multi-hop learning, and more.

Organization. Sec. 2 provides background on relational tables and predictive task specification. Sec. 3 introduces our central methodological contribution, a graph neural network approach to solving predictive tasks on relational data. Sec. 4 introduces RelBench, a new benchmark for relational tables, and standardized evaluation protocols. Sec. 5 outlines a landscape of new research opportunities for graph machine learning on relational data. Finally, Sec. 6 concludes by contextualizing our new framework within the tabular and graph machine learning literature.

2 Predictive Tasks on Relational Tables

This section outlines our problem scope: predictive tasks on relational tables. In the process, we rigorously define what we mean by relational tables, and how to specify predictive tasks on relational tables. This section focuses exclusively on the structure of data and tasks, laying the groundwork for Sec. 3, which presents our graph neural network-based modelling approach.

2.1 Relational Data

A Brief History. Relational tables and relational databases emerged in the 1970s as a means to standardize data retrieval and management [Codd, 1970]. As society digitized, relational databases came to fulfill a foundational purpose, and today are estimated to comprise 72% of the worlds data management and storage systems, according to DB-Engines, an initiative for monitoring database system usage [DB-Engines, 2023]. Whilst there is no single agreed upon definition of a relational database, three essential characteristics are shared in all cases (*cf.* Figure 1a):

1. Data is stored in multiple tables.
2. Each row in each table contains an *entity*, which possesses a unique primary key ID, along with multiple attributes stored as columns of the table.
3. One entity may refer to another entity using a foreign key, the primary key of another entity.

As well as a standardized storage system, relational databases typically come equipped with a powerful set of *relational operations*, which are used to manipulate and access data. Codd [1970] introduced 8 relational operations, including set operations such as taking the union of two tables, and other operations such as *joining* two tables based on their common attributes. Popular query languages such as SQL [Chamberlin and Boyce, 1974] provide commercial-grade implementations of a wide variety of relational operations. Next we formally define relational data, as suits our purposes.

Definition of Relational Databases. A relational database $(\mathcal{T}, \mathcal{L})$ is comprised of a collection of tables $\mathcal{T} = \{T_1, \dots, T_n\}$, and links between tables $\mathcal{L} \subseteq \mathcal{T} \times \mathcal{T}$ (*cf.* Figure 2a). 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** $\mathcal{K}_v = \{p_{v'} : v' \in T' \text{ and } (T, T') \in \mathcal{L}\}$, where $p_{v'}$ is the primary key of an entity in table T'.
3. **Attributes** x_v, holding the informational contents 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 primary key is TRANSACTIONID, foreign keys PRODUCTID and CUSTOMERID, one attribute, PRICE, and timestamp column TIMESTAMP. Similarly, the PRODUCTS table has primary key PRODUCTID, no foreign keys, attributes DESCRIPTION, IMAGE and SIZE, and no timestamp. The connection between foreign keys and primary keys is illustrated by black connecting lines.

In general, the attributes $x_v = (x_v^1, \dots, x_v^{d_T})$, contains multiple values, each belonging to a particular column. Critically, all entities in the same table have the same columns (values may be absent). Formally this is described by membership $x_v = (x_v^1, \dots, x_v^{d_T}) \in \mathcal{A}_T^1 \times \dots \times \mathcal{A}_T^{d_T}$, where $\mathcal{A}_T^i$ denotes column space i for table T (such as image space, or text space), and is shared between all entities $v \in T$.

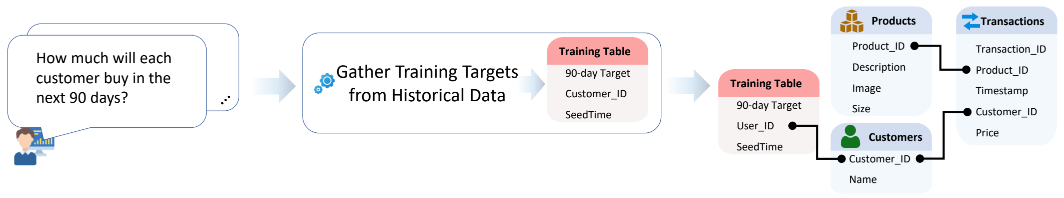
Fact and Dimension Tables. Tables are categorized into two types, *fact* or *dimension*, with complementary roles [Garcia-Molina et al., 2008]. Dimension tables provide contextual information such as biographical information, macro statistics (such as number of beds in a hospital), or immutable properties such as the size of a product (as in the PRODUCTS table in Figure 2a). Dimension tables tend to have relatively few rows, as it is limited to one per real-world object. Fact tables record interactions between other entities, such as all patient admissions to hospital, or each customer transaction (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 contain temporal information with a dedicated time column that denotes the time of appearance.

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 entity v. Furthermore, many *tasks* of interest involve forecasting future events. It is therefore essential that time is conferred a special status unlike other attributes. Our formulation, introduced in Sec. 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. This ensures that models do not leak information from the future during training, avoiding shortcut decision rules that achieve high training accuracy but fail at test time [Geirhos et al., 2020]. It also means that model-extracted are features automatically updated as new relational data is added.

2.2 Specifying Predictive Tasks on Relational Tables

The promise of machine learning on relational data is to inform decision making by, for instance, anticipating the response of a patient to treatment, or the future sales of a product. Here we explain how to formally specify a predictive task. The key concept is the notion of a *training table* (Fig. 3), a new table added to the relational database. The training table holds the ground truth labels, optional timestamps, and keys indicating which (combination of) entities each label is associated with.

Again, careful handling of time is crucial. Many important predictive tasks make predictions about future properties of entities, and are therefore inherently temporal. A model that aims to perform



(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.

predictions about the future must be trained on historical input features and historical prediction targets, which are all extracted from the given relational database. Thus, when the model is trained on a training example that was sampled from a specific time s in the past, it is of utmost importance to ensure that the model only sees the state of the database as it was before that time t.

Thus, the purpose of the training table is twofold. It is to hold historical training examples for each entity for which predictions should be made, including target value and foreign key(s) identifying the entity in question. And, in case of temporal tasks, it also needs to store the time each historical target *would have been predicted*, defining what database state the model is allowed to see when trained to make this prediction. As an example, if the model is trained to predict *how likely a customer is to churn in the next 7 days*, the training table will contain the binary churn label (1) for each customer and (2) for each historical week in the database. Each training example will have a timestamp, denoting the beginning of the respective week.

Predictive Task. We now formalize predictive task specification. Given a relational database $(\mathcal{T}, \mathcal{L})$, a predictive task is defined by a new relational database $(\mathcal{T}', \mathcal{L}')$. A new table T_{train} known as the *training table* is added to $\mathcal{T}$, creating $\mathcal{T}' = \mathcal{T} \cup \{T_{\text{train}}\}$. 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 $\mathcal{K}_v$ belonging to a set of existing tables $\mathcal{L}_{T_{\text{train}}} \subseteq \mathcal{T}$, (2) a timestamp t_v, and (3) the ground truth label itself y_v. The key links $\mathcal{L}_{T_{\text{train}}}$ specify the new set of table-level links $\mathcal{L}' = \mathcal{L} \cup \bigcup_{T' \in \mathcal{L}_{T_{\text{train}}}} \{T, T'\}$. In contrast to tabular learning settings, the training table does *not* contain input data x_v. The timestamp is used to avoid temporal information leakage. A training table entity v with timestamp t_v will only be permitted to receive information from entities u with timestamp $t_u \leq t_v$ (see Sec. 3.3 for full details).

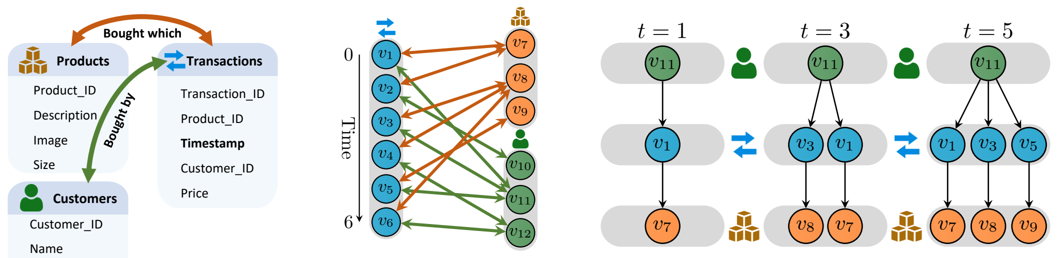
This training table formulation can model a wide range of predictive tasks on relational databases:

- For **Node-level prediction** tasks (multi-class classification, multi-label classification, regression) training table have columns (ENTITY_ID, LABEL, TIME)
- **Link prediction** task training tables have columns (LEFT_ENTITY_ID, RIGHT_ENTITY_ID, LABEL, TIME)
- It can model **temporal** and **static** prediction tasks (TIME is dropped for static), where temporal tasks make predictions about the future (and require a seed time), while static tasks impute missing values.

Training Table Generation. In practice, training tables can 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 s as timestamp for the targets gathered in each step.

3 Predictive Tasks as Graph Representation Learning Problems

Here, we formulate a generic machine learning architecture based on Graph Neural Networks, which solves predictive tasks on relational databases. The following section will first introduce three



(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.

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 graph neural network blueprint for learning on relational databases in an end-to-end framework (*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 $\mathcal{L}^{-1} = \{(T, T') \mid (T', T) \in \mathcal{L}\}$ 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 relational learning 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 $(\mathcal{T}, \mathcal{R})$, 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

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

i.e. the entity-level pairs that arise from the primary-foreign key relationships in the database. Type mapping functions $\phi : \mathcal{V} \to \mathcal{T}$ and $\psi : \mathcal{E} \to \mathcal{R}$ are given by which table or table-link the entity or link belongs to. 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 $(v_1, v_2) \in \mathcal{E}$.

We equip the relational entity graph with the following additional information:

- **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 renders the graph *temporal*. 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.

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 $G' = (\mathcal{V}', \mathcal{E}')$
  $\mathcal{V}'_0 \leftarrow \{v_0\}, \quad \mathcal{E}'_0 \leftarrow \emptyset$
  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
  $\mathcal{V}' \leftarrow \bigcup_{i=1}^{L} \mathcal{V}'_i, \quad \mathcal{E}' \leftarrow \bigcup_{i=1}^{L} \mathcal{E}'_i$
```

- **Embedding vectors** $\mathbf{h}_v \in \mathbb{R}^{d_{\phi(v)}}$ for each $v \in \mathcal{V}$, which contains an embedding vector for each graph node. Initial embeddings are obtained via multi-modal feature encoders as described in 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 rows foreign key entry links to the primary key entry of the other nodes 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.

Besides the need for time consistency, computational graphs also ensure scalability of our proposed approach through sampling opportunities. Today's relational databases can contain billions of rows, which makes efficient mini-batch sampling mandatory. In practice, we can create computational graphs through neighbor sampling [Hamilton et al., 2017] while taking care that temporal constraints of samples are met [Wang et al., 2021]. Specifically, 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 $G' = (\mathcal{V}', \mathcal{E}')$ 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' 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.

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 **feature encoders** that encode table column raw data into initial node embeddings $\mathbf{h}_v^{(0)}$. Generic feature encoders are described in 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 an optimal model for the given task.

3.4.1 Relational-Temporal Message Passing

This section introduces a generic framework for heterogeneous message passing GNNs on relational entity graphs as defined in Sec. 3.2. A message passing operator in the given relational framework needs to respect the heterogeneous nature as well as the temporal properties of the graph. This is ensured by filtering nodes based on types and time. Thus, we briefly introduce heterogeneous message passing before we turn to our temporal message passing.

Heterogeneous Message Passing. *Message-Passing Graph Neural Networks (MP-GNNs)* [Gilmer et al., 2017, Fey and 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:

$$\mathbf{h}_v^{(i+1)} = f(\mathbf{h}_v^{(i)}, \{\!\{ g(\mathbf{h}_w^{(i)}) \mid w \in \mathcal{N}(v) \}\!\}), \tag{2}$$

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. 2, adding an aggregation over all incoming edge types to learn distinct message types:

$$\mathbf{h}_v^{(i+1)} = f_{\phi(v)}\left(\mathbf{h}_v^{(i)}, \left\{\!\!\left\{ f_R(\{\!\{ g_R(\mathbf{h}_w^{(i)}) \mid w \in \mathcal{N}_R(v) \}\!\}) \,\middle|\, \forall R = (T, \phi(v)) \in \mathcal{R} \right\}\!\!\right\}\right), \tag{3}$$

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 and 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. 3, where we additionally filter R-specific neighborhoods based on their timestamp, *i.e.* replace $\mathcal{N}_R(v)$ with

$$\mathcal{N}_R^{\leq t}(v) = \{w \in \mathcal{V} \mid (w, v) \in \mathcal{E}, \psi(w, v) = R, \text{and } \tau(w) \leq t\}, \tag{4}$$

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.

3.4.2 Predict with Model Heads

The model described so far is task-agnostic and simply propagates information through the relational entity graph to produce generic 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 $\{(p, \mathcal{K}, t, y)_i\}_{i=1}^N$ (*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 is a function that maps node-level embeddings $\mathbf{h}_v^{(L)}$ to a prediction $\hat{y}$, *i.e.*

$$f : \mathbb{R}^{d_v} \to \mathbb{R}^d, \qquad f : \mathbf{h}_v^{(L)} \mapsto \hat{y}. \tag{5}$$

Link-level Model Head. Similarly, we can define a link-level model head for training examples $\{(p, \mathcal{K}, t, y)_i\}_{i=1}^N$ 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.*

$$f : \mathbb{R}^{d_{v_1}} \times \mathbb{R}^{d_{v_2}} \to \mathbb{R}^d, \qquad f : (\mathbf{h}_{v_1}^{(L)}, \mathbf{h}_{v_2}^{(L)}) \mapsto \hat{y}. \tag{6}$$

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. In the first version, RelBench provides node-level tasks only.

3.4.3 Multi-Modal Node Encoders

The final piece of the pipeline is the initial entity-level node features $\mathbf{h}_v^{(0)}$, which are extracted from attributes $x_v = (x_v^1, \ldots, x_v^{d_T}) \in \mathcal{A}_T^1 \times \ldots \times \mathcal{A}_T^{d_T}$ using modality specific encoders. An encoder $f_{\mathcal{A}_T^i}$ is defined for each column, as each column contains a fixed data modality, such as images, text, categorical, or numerical values. Importantly, this supports pre-trained foundation models for text, image and other common modalities. The encoded attributes are aggregated into a single unified initial node embeddings $\mathbf{h}_v^{(0)} = \{\!\{ f_{\mathcal{A}_T^1}(x_v^1), \ldots, f_{\mathcal{A}_T^{d_T}}(x_v^{d_T}) \}\!\}$ of an entity, where $\{\!\{\cdot\}\!\}$ is an aggregation function such as concatenation, and does not have to be the same as the GNN aggregation in Sec. 3.4.1. In practice, we rely on the wide range of encoders implemented in PyTorch Frame [Hu et al., 2023] which includes, for instance, state-of-the-art text encoders from OpenAI and HuggingFace.

3.5 Discussion

The neural network architecture presented in this blueprint is end-to-end trainable directly on data in relational databases. This approach supports a wide range of tasks, such as classification, regression or link prediction in a unified way, defined by an easy to obtain training table. It learns to solve tasks without requiring manual feature engineering, as typical in tabular learning. Instead, operations that are otherwise done manually, such as SQL `JOIN+AGGREGATE` operations, are learned by the GNN. More than simply replacing SQL operations, the GNN message and aggregation steps *exactly match* the functional form of SQL `JOIN+AGGREGATE` operations. In other words, the GNN is an exact *neural version* of SQL `JOIN+AGGREGATE` operations. We believe this is another important reason why message passing-based architectures are a natural learned replacement for hand-engineered features on relational tables.

4 RelBench: A Benchmark for Relational Deep Learning

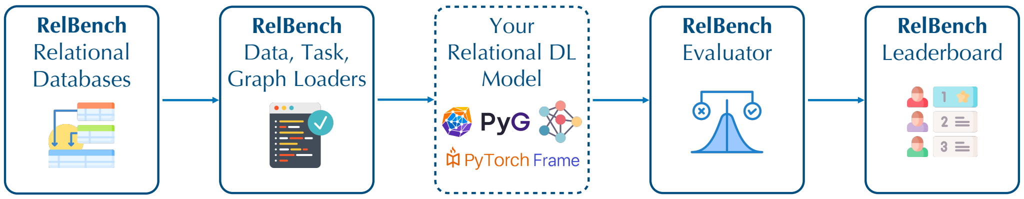


Figure 5: **Overview of RelBench.** RelBench enables training and evaluation of machine learning models on relational data. RelBench supports deep learning framework agnostic data loading, task specification, standardized data splitting, and transforming data into graph format. RelBench provides standardized evaluation metric computations, and a leaderboard for tracking progress. We additionally provide example training scripts built using PyTorch Geometric and PyTorch Frame.

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 functionalities. In its beta release[1][2], we announce the first two real-world relational databases, each with two curated predictive tasks.

In the subsequent sections (Sec. 4.3 and 4.4), 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. 4.1.

4.1 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 PyG 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. We refer the readers to the code repository for a more detailed understanding of RELBENCH. Here we demonstrate the core functionality.

To load a relational database, simply do:

```python
from relbench.datasets import get_dataset
dataset = get_dataset(name="rel-amazon")
```

It will load the relational tables and process it into a standardized format. Next, to load the predictive task and the relevant training tables, do:

```python
task = dataset.get_task("rel-amazon-ltv")
task.train_table, task.val_table, task.test_table # training/validation/testing tables
```

It automatically constructs the training table for the relevant predictive task. Next, after the user trains the machine learning model, the user can use RELBENCH standardized evaluator:

```python
task.evaluate(pred)
```

4.2 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.

4.3 `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

[1]Website: https://relbench.stanford.edu
[2]Package: https://github.com/snap-stanford/relbench

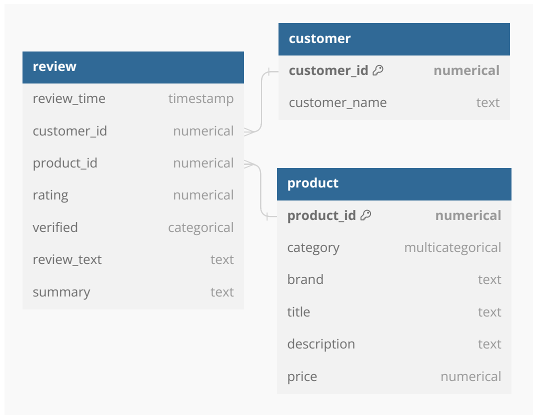


Figure 6: `rel-amazon` contains two dimension tables (customers and products) and one fact table (reviews). Each review has a customer and a product foreign key.

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.

4.3.1 `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 sum of prices of the 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-\$33,858.4 in the given time window in the training table.

Evaluation metric: Mean Absolute Error (MAE).

4.3.2 `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).

4.4 `rel-stackex`: Stack exchange question-and-answer website database

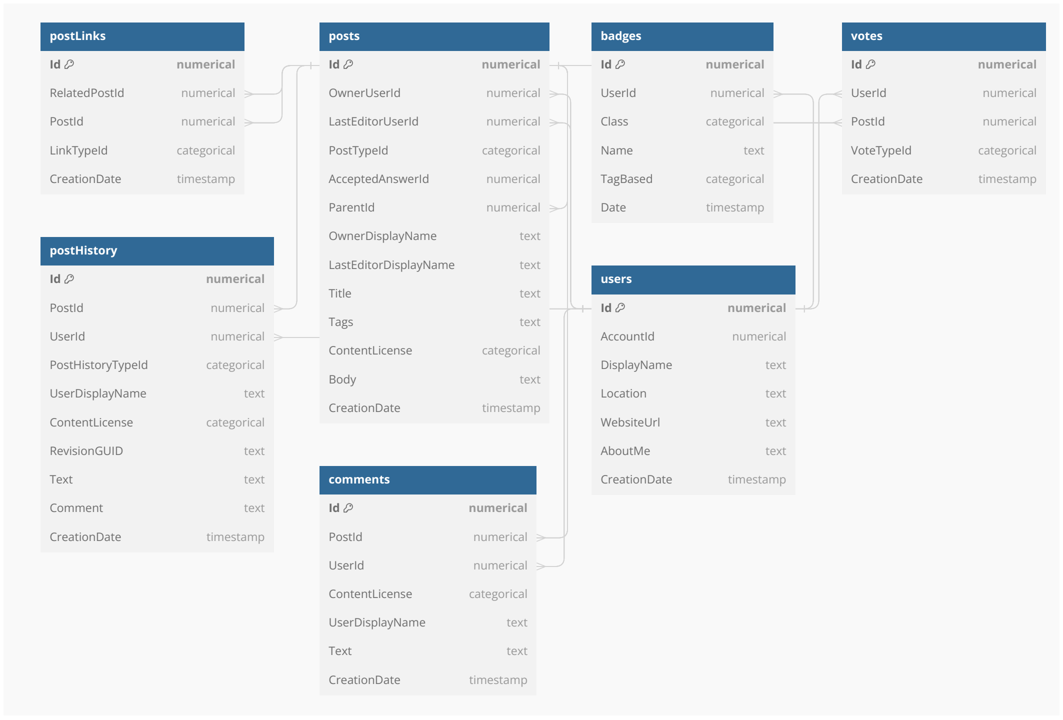


Figure 7: Entity relational diagrams of `Stack-Exchange`.

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.

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.

4.4.1 `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).

4.4.2 `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: Regression. The target ranges from 0-52 number of upvotes in the given time window in the training table.

Evaluation metric: Mean Absolute Error (MAE).

5 A New Program for Graph Representation Learning

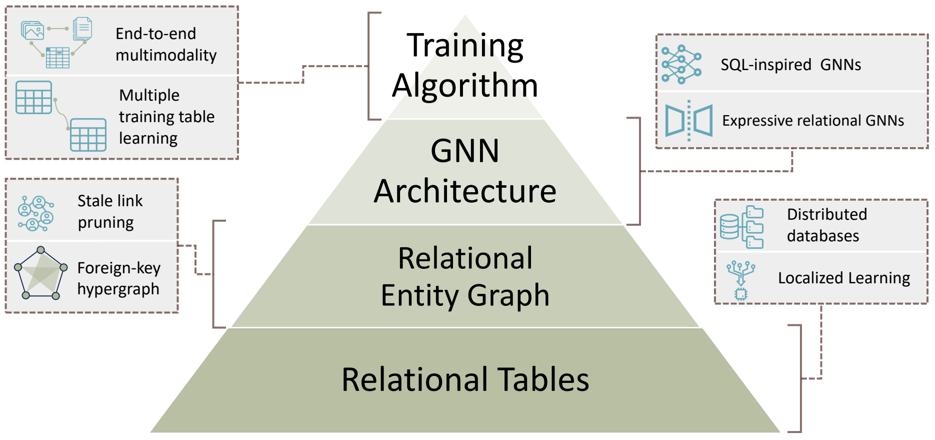


Figure 8: Relational Deep Learning brings new challenges at all levels of the machine learning stack.

Developing Relational Deep Learning requires a new research program in graph representation learning on relational data. There are opportunities at all levels of the research stack, including (pre-)training methods, GNN architectures, multimodality, new graph formulations, and scaling to large distributed relational databases as occur in many practical settings. Here we discuss several promising aspects of this research program, aiming to stimulate the interest of the graph machine learning community.

5.1 Scaling Relational Deep Learning

Relational databases are often vast, with information distributed across many servers with constrained communication. However, relational data has a non-typical graph structure which may help scale Relational Deep Learning more efficiently.

Distributed Training on Relational Data. Existing distributed machine learning techniques often assume that each server contains data of the same type. Relational data on the other hand, naturally partitions in to pieces, bringing new challenges depending on the partitioning technique used. *Horizontal partitioning*, known as sharding, is the most common approach. It creates database shards by splitting tables row-wise according to some criterion (e.g., all customer with zipcode in a given range). In this case, a table containing a customers personal record may lie on a distinct server from the table recording recent purchase activity, leading to communication bottlenecks when attempting to train models by sensing messages between purchases and customer records. Less common, but also possible, is vertical splitting. Different splitting options suggests an opportunity to (a) develop specialized distributed learning methods that exploit the vertical or horizontal partitioning, and (b) design further storage arrangements that may be more suited to Relational Deep Learning. The question of graph partitioning arises in all large-scale graph machine learning settings, however in this case we are fortunate to have non-typical graph structure (i.e., it follows the schema) which makes it easier to find favourable partitions.

Localized Learning. For many predictive tasks it is neither feasible (due to database size) nor desirable (due to task narrowness) to propagate GNN messages across the entire graph during training. In such cases, sampling schemes that preserve locality by avoiding exponential growth in GNN receptive field are needed. This is easily addressed in cases with prior knowledge on the relevant entities. How to scale to deep models that remain biased models towards local computation in cases with no prior knowledge remains an interesting open question.

5.2 Building Graphs from Relational Data

An essential ingredient of Relational Deep Learning is the use of an individual entity and relation-level graph on which to apply inter-entity message passing to learn entity embeddings based on relations to other entities. In Sec. 3.2 we introduced one such graph, the *relational entity graph*, a general procedure for viewing any relational database as a graph. 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. It is this property that we expect potential alternative graph designs to share. Beyond this stipulation, many creative graph choices are possible, and we discuss some possibilities here.

Foreign-key Hypergraph. Fact tables often contain entities with a fixed foreign-key pattern (*e.g.*, in `rel-amazon` 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. 5.3).

Stale Link Pruning. Entities that have been active for a long time may have a lot of links to other entities. Many of these links may be *stale*, or uninformative, for certain tasks. For example, the purchasing patterns of a longtime customer during childhood are likely to be less relevant to their purchasing patterns in adulthood. Links that are stale for a certain task may hurt predictive power by obfuscating true predictive signals, and reduce model efficiency due to processing uninformative data. This situation calls for careful stale link and entity handling to focus on relevant information. Promising methods may include pruning or preaggregating stale links. More generally, how to deal with more gradual distribution drift over time is an open question.

5.3 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. Here we discuss several concrete directions for designing new architectures.

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. This suggests that GNNs for relational data should be designed to be capable of learning expressive decision rules over n-partite graphs. 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 `rel-amazon` 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). How to integrate prior knowledge of the graph structure of $(\mathcal{T}, \mathcal{R})$ into GNNs that operate on an entity-level graph (the relational entity graph) remains an open question. These two examples serve only to illustrate the possibilities for architecture design based on the structure of relational entity graphs. Many other structural properties of relational data may lead to innovative new expressive GNN architectures.

Query Language Inspired Models. SQL operations are known to be extremely powerful operations for manipulating relational data. Their weakness is that they do not have differentiable parameters, making end-to-end learnability impossible. Despite this, 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. There are many choices of `AGGREGATE` operation such as `SUM`, `MEAN` and `COUNT`. This process *directly* mirrors GNN computations of messages from neighboring nodes, followed by message aggregation. In other words, GNNs can be thought of as a neural version of SQL `JOIN+AGGREGATE` operations. This suggests that an opportunity for powerful new neural network architectures by designing differentiable computation blocks that algorithmically align [Xu et al., 2020] to existing SQL operations that are known to be useful.

New Message Passing Schemes. Beyond expressivity, new architectures may also improve information propagation between entities. For instance, collaborative filtering methods enhance predictions by identify entities with similar behavior patterns, customers with similar purchase history. However, in the relational entity graph, the twp 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 (2-hops from v_1 to product, and 2-hops from product to v_1). New message passing schemes that do multiple hops or directly connect customers (more generally, entities) with similar behavior patterns may more effectively propagate key information through the model. As well as new message passing schemes, there is also opportunity for new message aggregation methods. One possibility is order dependent aggregation, that combines messages in a time-dependent way, as explored by Yang et al. [2022]. Another is schema dependent aggregation, that combines messages based on what part of the schema graph the messages are arriving from.

5.4 Training Techniques for Relational Data

By its nature, relational data contains highly overlapping predictive signals and tasks. This interconnectedness of data and tasks is a big opportunity for new neural network training methods that maximally take advantage of this interconnectedness to identify useful predictive signals. This section discusses several such opportunities.

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 in relational databases often have attributes covering multiple modalities (*e.g.,* products come with both images and description). The Relational Deep Learning blueprint first extracting entity-level features, which are used as initial node-features for the GNN model. In RELBENCH, this multimodal entity-level feature extraction is handled by using state-of-the-art pre-trained models using the PyTorch Frame library to pre-extract features. 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.

To facilitate exploration of joint entity-level and graph-level modelling, RELBENCH also provides the option to load raw data, to allow researchers to experiment with different feature-extraction methods.

Foundation Models and Data Type Encoders. In practice, new predictive tasks on relational data are often specified on-the-fly, with fast responses required. Such situations preclude costly model training from scratch, instead requiring powerful and generic pre-trained models. Self-supervised labels for model pre-training can be mined from historical data, just as with training table construction. However, techniques for automatically deciding which labels to mine remains unexplored. Another desirable property of pre-trained models is that they are *inductive*, so they can be applied to entirely new relational databases out-of-the-box. This presents a challenges in how to deal with unseen column types and relations between tables. Such flexibility is needed in order to move towards foundation models for relational databases. More broadly, how to build column encoders is an important question. As well as distribution shifts as mentioned in the previous paragraph, there are also decisions on when to share column encoders (should two image columns use the same image encoder?), as well as special data types such as static time intervals (*e.g.*, to describe the time period an employee worked at a company, or the time period in which a building project was conducted). Special data types may require specialized encoder choices, and possibly even deeper integration into the neural network computation graph. How best to aggregate of cross-modal information into a single fused embedding is another question for exploration.

6 Related Work

Statistical Relational Learning. Since the foundation of the field of AI, 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 and 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 and 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, since both approaches operate on data with rich relational structure, and both approaches integrate relational structure into the model design. Of course, there are important distinctions between the two methods too, such as the natural scalability of graph neural network-based methods, and that Relational Deep Learning does not rely on first-order logic to describe data, allowing broad applicability to relations that are hard to fit into this form.

Tabular Machine Learning. Tree based methods, notably XGBoost [Chen and 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 and Pfister, 2021], but have struggled to clearly establish dominance over tree-based methods [Shwartz-Ziv and Armon, 2022]. The vast majority of tabular machine learning focuses on the single table setting, which we argue forgoes use of the rich interconnections between relational data. As such, it does not address the key problem, which is how to get the data from a multi-table to a single table representation. Recently, a nascent body of work has begun to consider multiple tables. For instance, Zhu et al. [2023] pre-train tabular Transformers that generalize to new tables with unseen columns.

Knowledge Graph Embedding. Knowledge graphs store relational data, and highly scalable knowledge graph embeddings methods have been developed over the last decade to embed data into spaces whose geometry reflects the relational struture [Bordes et al., 2013, Wang et al., 2014, 2017]. Whilst also dealing with relational data, this literature differs from this present work in the task being solved. The key task of knowledge graph embeddings is to predict missing entities (Q: Who was Yann LeCun's postdoc advisor? A: Geoffrey Hinton) or relations (Q: Did Geoffrey Hinton win a Turing Award? A: Yes). To assist in such *completion* tasks, knowledge graph methods learn an embedding space with the goal of exactly preserving the relation semantics between entities. This is different from Relational Deep Learning, which aims to make *predictions* about entities, or groups of entities. Because of this, Relational Deep Learning seeks to leverage relations to learn entity representations, but does not need to learn an embedding space that perfectly preserves all relation semantics. This gives more freedom and flexibility to our models, which may discard certain relational information it finds unhelpful. Nonetheless, adopting ideas from knowledge graph embedding may yet be fruitful.

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, such as heterogeneous message passing. However, our aim is to move beyond the conceptual level, and clearly establish deep learning on relational data as a subfield of machine learning. Accordingly, 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.

7 Conclusion

A large proportion of the worlds data is natively stored in relational tables. Fully exploiting the rich signals in relational data therefore has the potential to rewrite what problems computing can solve. We believe that Relational Deep Learning will make it possible to achieve superior performance on various prediction problems spanning the breadth of human activity, leading to considerable improvements in automated decision making. There is currently a great scientific opportunity to develop the field of Relational Deep Learning, and further refine this vision.

This paper serves as a road map in this pursuit. We introduce a blueprint for a neural network architecture that directly processes relational data by casting predictive tasks as graph representation learning problems. In Sec. 5 we discuss the many new challenges and opportunities this presents for the graph machine learning community. To facilitate research, we introduce RelBench, a set of benchmark datasets, and a Python package for data loading, and model evaluation.

References

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

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems*, 26, 2013.

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

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

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

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

DB-Engines. DBMS popularity broken down by database model, 2023. Available: https://db-engines.com/en/ranking_categories.

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

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

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

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

Lise Getoor, Nir Friedman, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. *Relational data mining*, pages 307–335, 2001.

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

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf.

Weihua Hu, Matthias Fey, Yiwen Yuan, Zecheng Zhang, Akihiro Nitta, Kaidi Cao, and Vid Kocijan. PyTorch Frame: A Deep Learning Framework for Tabular Data, October 2023. URL https://github.com/pyg-team/pytorch-frame.

Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*, page 2704–2710, 2020.

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. 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.

Nada Lavrac and Saso Dzeroski. Inductive logic programming. In *WLP*, pages 146–160. Springer, 1994.

Marvin Minsky. 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/.

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

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web*, pages 593–607, Cham, 2018. Springer International Publishing.

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

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

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

Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12): 2724–2743, 2017.

Yiwei Wang, Yujun Cai, Yuxuan Liang, Henghui Ding, Changhu Wang, and Bryan Hooi. Time-aware neighbor sampling for temporal graph networks. In *arXiv pre-print*, 2021.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI conference on artificial intelligence*, volume 28, 2014.

Keyulu Xu, Jingling Li, Mozhi Zhang, Simon S Du, Ken-ichi Kawarabayashi, and Stefanie Jegelka. What can neural networks reason about? In *International Conference on Learning Representations (ICLR)*, 2020.

Zhen Yang, Ming Ding, Bin Xu, Hongxia Yang, and Jie Tang. Stam: A spatiotemporal aggregation method for graph neural network-based recommendation. In *Proceedings of the ACM Web Conference 2022*, pages 3217–3228, 2022.

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

Bohang Zhang, Shengjie Luo, Liwei Wang, and Di He. Rethinking the expressive power of gnns via graph biconnectivity. *arXiv preprint arXiv:2301.09505*, 2023.

Bingzhao Zhu, Xingjian Shi, Nick Erickson, Mu Li, George Karypis, and Mahsa Shoaran. Xtab: Cross-table pretraining for tabular transformers. *arXiv preprint arXiv:2305.06090*, 2023.