Relational Deep Learning: Graph Representation Learning on Relational Databases

Joshua Robinson[1]*, **Rishabh Ranjan**[1]*, **Weihua Hu**[2]*, **Kexin Huang**[1]*,
Jiaqi Han[1], **Alejandro Dobles**[1], **Matthias Fey**[2], **Jan E. Lenssen**[2,3],
Yiwen Yuan[2], **Zecheng Zhang**[2], **Xinwei He**[2], **Jure Leskovec**[1,2]
[1]Stanford University [2]Kumo.AI [3]Max Planck Institute for Informatics

https://relbench.stanford.edu

Abstract

Much of the world's most valuable data is stored in relational databases, where data is organized into tables connected by primary-foreign key relationships. Building machine learning models on this data is challenging because existing algorithms cannot directly learn from multiple connected tables. Current methods require manual feature engineering, which involves joining and aggregating tables into a single format, a labor-intensive and error-prone process. We introduce *Relational Deep Learning (RDL)*, an end-to-end learning framework that eliminates the need for manual feature engineering by representing relational databases as temporal, heterogeneous graphs. In this representation, rows become nodes, and primary-foreign key links define edges. Graph Neural Networks (GNNs) are then used to learn representations from all available data. We benchmark RDL on RELBENCH, evaluating 30 predictive tasks across seven relational databases, and demonstrate superior performance compared to traditional methods. In a user study, RDL significantly outperforms an experienced data scientist's manual feature engineering approach, reducing human effort by more than 90%. These results highlight the potential of deep learning for predictive tasks in relational databases.

1 Introduction

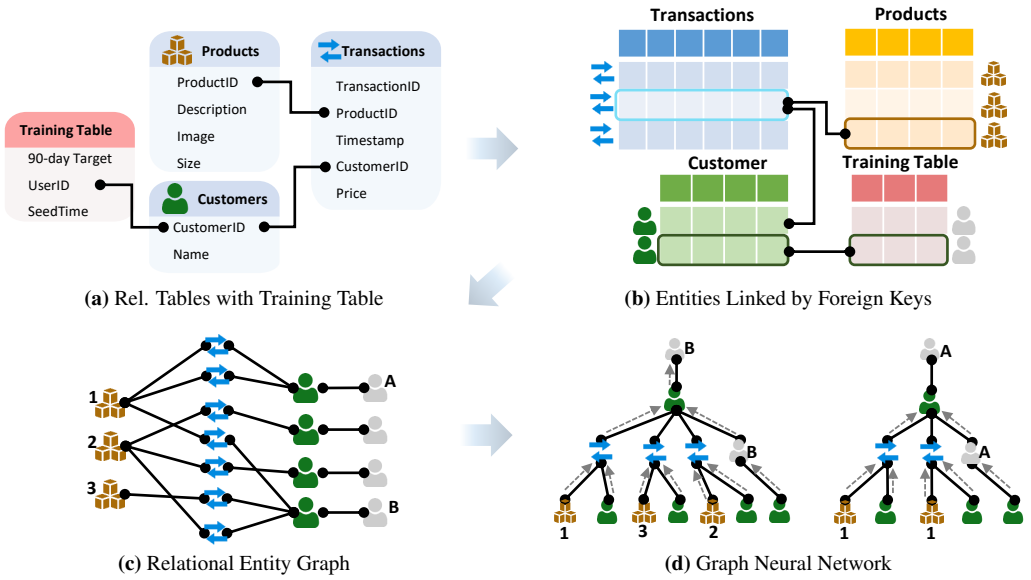
The information age is fueled by data stored in relational databases, which are foundational to nearly all modern technology stacks. These databases store data across multiple tables, linked by primary and foreign keys, and are managed using query languages like SQL (Codd, 1970; Chamberlin and Boyce, 1974). As a result, they are at the core of systems in sectors like e-commerce, social media, healthcare, and scientific repositories (Johnson *et al.*, 2016; PubMed, 1996).

Many predictive tasks over relational databases hold significant implications for decision making. For example, a hospital might predict patient discharge risk, or a company may forecast product sales. Each of these tasks relies on rich relational schemas, and machine learning models are often built from this data (Kaggle, 2022).

However, existing machine learning paradigms, especially tabular learning, cannot directly operate on relational data. Instead, a manual feature engineering process is often required, where data from multiple tables is aggregated into a single table format for learning. For instance, in an e-commerce schema, a data scientist might extract features like "number of purchases in the last 30 days" to predict future customer behavior. These features are manually constructed and stored in a single table for training tabular models.

*Equal contribution, order chosen randomly. First authors may swap the ordering for professional purposes.

3rd Table Representation Learning Workshop at NeurIPS 2024.



(a) Rel. Tables with Training Table

(b) Entities Linked by Foreign Keys

(c) Relational Entity Graph

(d) Graph Neural Network

Figure 1: **Relational Deep Learning Pipeline. (a)** Relational tables with a predictive task generate a training table containing supervised labels. **(b)** Entities are linked by foreign-primary key relations. **(c)** The data forms a *Relational Entity Graph*, with nodes for each entity and edges from key links. **(d)** Features are extracted for each entity, and a GNN computes embeddings. A model head produces predictions, and errors are backpropagated.

This manual approach has significant limitations: it is labor-intensive, suboptimal, and often fails to capture fine-grained signals. Moreover, temporal features need frequent recomputation, leading to additional computational cost and potential time-leakage bugs (Kapoor and Narayanan, 2023). These challenges are similar to those faced in early computer vision, where hand-engineered features were eventually replaced by end-to-end deep learning systems (He *et al.*, 2016; Russakovsky *et al.*, 2015).

Here we introduce *Relational Deep Learning (RDL)*, a framework for end-to-end deep learning on relational databases. RDL fully leverages relational data by representing it as a heterogeneous *Relational Entity Graph*, where rows become nodes, columns become node features, and primary-foreign key links form edges. Graph Neural Networks (GNNs) (Gilmer *et al.*, 2017; Hamilton *et al.*, 2017) are then applied to learn predictive models directly from the graph structure.

RDL consists of four main steps (Fig. 1): (1) A *training table* containing labels is computed from historic data, (2) entity-level features are extracted as node features, (3) node embeddings are learned via a GNN that exchanges information across primary-foreign key links, and (4) a task-specific model produces predictions, with errors backpropagated for optimization.

RDL naturally handles temporal data by ensuring entities only receive messages from earlier timestamps, preventing information leakage. This allows for efficient model updating and avoids common time-travel bugs.

We evaluate RDL on 30 tasks across 7 databases in the RELBENCH benchmark. Our method outperforms baselines, including a strong "data scientist" approach where features are manually engineered for each task. RDL models match or exceed these models in accuracy while reducing human labor by 96% and lines of code by 94%. This demonstrates RDL's promise as the first end-to-end deep learning solution for relational data, offering significant improvements in both accuracy and efficiency.

2 Problem Formulation

This section defines predictive tasks on relational tables, focusing on data structure and task specification. We lay the groundwork for Section 3, where we present our GNN-based approach.

A relational database $(\mathcal{T}, \mathcal{L})$ consists of tables $\mathcal{T} = T_1, \ldots, T_n$, and links $\mathcal{L} \subseteq \mathcal{T} \times \mathcal{T}$, which connect tables through primary-foreign key relations (Fig. 1a). Each table T is a set of rows, or entities v, with a primary key, foreign keys, attributes, and an optional timestamp. For instance, in Fig. 1a, TRANSACTIONS has a primary key (TRANSACTIONID), two foreign keys, attributes, and a timestamp.

Attributes are tuples of values $x_v = (x_v^1, \ldots, x_v^{d_T})$ shared across entities in the same table. For example, PRODUCTS contains text, image, and numerical attributes, each processed by specific encoders (Sec. 3.4.3).

Predictive tasks often involve forecasting future events. Ground truth labels for training are generated from historical data (e.g., summing customer purchases over a future period). To hold these labels, we introduce a *training table* T_{train}, where each row has a foreign key, a timestamp, and a label.

The training table links to the main database via foreign keys, and timestamps ensure temporal consistency, preventing data leakage. This setup supports node-level, link prediction, and both temporal and static tasks, providing a versatile framework for various machine learning problems on relational data.

3 Methods

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: (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.*, 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

$$\mathcal{E} = \{(v_1, v_2) \in \mathcal{V} \times \mathcal{V} \mid p_{v_2} \in \mathcal{K}_{v_1} \text{ or } p_{v_1} \in \mathcal{K}_{v_v}\}, \tag{3.1}$$

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: $\tau : v \mapsto t_v$, 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.

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, 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).

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 and Lenssen, 2019; Schlichtkrull *et al.*, 2018; Hu *et al.*, 2020) and extend it by a temporal filtering mechanism. Given a relational entity graph, initial node embeddings $\{\mathbf{h}_v^{(0)}\}_{v \in \mathcal{V}}$ and an example specific *seed time* $t \in \mathbb{R}$, 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 $\{(\mathcal{K}, t, y)_i\}_{i=1}^N$, 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.*

$$f : \mathbb{R}^{d_v} \to \mathbb{R}^d, \qquad f : \mathbf{h}_v^{(L)} \mapsto \hat{y}. \tag{3.2}$$

Table 1: Entity classification results (AUROC, higher is better) on RELBENCH. Best values are in bold. See Table 4 in Appendix B for standard deviations.

Dataset	Task	Split	LightGBM	RDL	Rel. Gain of RDL
rel-amazon	user-churn	Val	52.05	**70.45**	35.35 %
		Test	52.22	**70.42**	34.86 %
	item-churn	Val	62.39	**82.39**	32.06 %
		Test	62.54	**82.81**	32.40 %
rel-avito	user-visits	Val	53.31	**69.65**	30.66 %
		Test	53.05	**66.20**	24.78 %
	user-clicks	Val	55.63	**64.73**	16.35 %
		Test	53.60	**65.90**	22.96 %
rel-event	user-repeat	Val	67.76	**71.25**	5.15 %
		Test	68.04	**76.89**	13.02 %
	user-ignore	Val	87.96	**91.70**	4.25 %
		Test	79.93	**81.62**	2.12 %
rel-f1	driver-dnf	Val	68.42	**71.36**	4.31 %
		Test	68.56	**72.62**	5.93 %
	driver-top3	Val	67.76	**77.64**	14.57 %
		Test	73.92	**75.54**	2.20 %
rel-hm	user-churn	Val	56.05	**70.42**	25.63 %
		Test	55.21	**69.88**	26.59 %
rel-stack	user-engagement	Val	65.12	**90.21**	38.53 %
		Test	63.39	**90.59**	42.91 %
	user-badge	Val	65.39	**89.86**	37.43 %
		Test	63.43	**88.86**	40.08 %
rel-trial	study-outcome	Val	**68.30**	68.18	−0.19 %
		Test	**70.09**	68.60	−2.13 %
Average		Val	64.18	**76.49**	20.34 %
		Test	63.66	**75.83**	20.48 %

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.*

$$f : \mathbb{R}^{d_{v_1}} \times \mathbb{R}^{d_{v_2}} \to \mathbb{R}^d, \qquad f : (\mathbf{h}_{v_1}^{(L)}, \mathbf{h}_{v_2}^{(L)}) \mapsto \hat{y}. \tag{3.3}$$

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 Experiments

We evaluate RDL on RELBENCH. Tasks are grouped into three task types: entity classification (Section 4.1), entity regression (Section 4.2), and entity link prediction (Section 4.3). Tasks differ significantly in the number of train/val/test entities, number of unique entities (the same entity may appear multiple times at different timestamps), and the proportion of test entities seen during training. Note this is not data leakage, since entity predictions are timestamp dependent, and can change over time. Tasks with no overlap are pure inductive tasks, whilst other tasks are (partially) transductive.

4.1 Entity Classification

The first task type is entity-level classification. The task is to predict binary labels of a given entity at a given seed time. We use the ROC-AUC (Hanley and McNeil, 1983) metric for evaluation (higher is better). We compare to a LightGBM classifier baseline over the raw entity table features. Note that here only information from the single entity table is used.

Table 2: Entity regression results (MAE, lower is better) on RELBENCH. Best values are in bold. See Table 5 in Appendix B for standard deviations.

Dataset	Task	Split	Global Zero	Global Mean	Global Median	Entity Mean	Entity Median	LightGBM	RDL	Rel. Gain of RDL
rel-amazon	user-ltv	Val	14.141	20.740	14.141	17.685	15.978	14.141	**12.132**	14.21 %
		Test	16.783	22.121	16.783	19.055	17.423	16.783	**14.313**	14.72 %
	item-ltv	Val	72.096	78.110	59.471	80.466	68.922	55.741	**45.140**	19.02 %
		Test	77.126	81.852	64.234	78.423	66.436	60.569	**50.053**	17.36 %
rel-avito	ad-ctr	Val	0.048	0.048	0.040	0.044	0.044	0.037	**0.037**	2.21 %
		Test	0.052	0.051	0.043	0.046	0.046	**0.041**	0.041	−0.18 %
rel-event	user-attendance	Val	0.262	0.457	0.262	0.296	0.268	0.262	**0.255**	2.65 %
		Test	0.264	0.470	0.264	0.304	0.269	0.264	**0.258**	1.97 %
rel-f1	driver-position	Val	11.083	4.334	4.136	7.181	7.114	3.450	**3.193**	7.44 %
		Test	11.926	4.513	4.399	8.501	8.519	4.170	**4.022**	3.56 %
rel-hm	item-sales	Val	0.086	0.142	0.086	0.117	0.086	0.086	**0.065**	24.50 %
		Test	0.076	0.134	0.076	0.111	0.078	0.076	**0.056**	26.90 %
rel-stack	post-votes	Val	0.062	0.146	0.062	0.102	0.064	0.062	**0.059**	4.19 %
		Test	0.068	0.149	0.068	0.106	0.069	0.068	**0.065**	4.11 %
rel-trial	study-adverse	Val	57.083	75.008	56.786	57.083	57.083	**45.774**	46.290	−1.13 %
		Test	57.930	73.781	57.533	57.930	57.930	**44.011**	44.473	−1.05 %
	site-success	Val	0.475	0.462	0.475	0.447	0.450	0.417	**0.401**	3.87 %
		Test	0.462	0.468	0.462	0.448	0.441	0.425	**0.400**	5.86 %
Average		Val	17.260	19.939	15.051	18.158	16.668	13.330	**11.952**	8.55 %
		Test	18.299	20.393	15.985	18.325	16.801	14.045	**12.631**	8.14 %

Experimental results. Results are given in Table 1, with RDL outperforming or matching baselines in all cases. Notably, LightGBM achieves similar performance to RDL on the `study-outcome` task from `rel-trial`. This task has extremely rich features in the target table (28 columns total), giving the LightGBM many potentially useful features even without feature engineering. It is an interesting research question how to design RDL models better able to extract these features and unify them with cross-table information in order to outperform the LightGBM model on this dataset.

4.2 Entity Regression

Entity-level regression tasks involve predicting numerical labels of an entity at a given seed time. We use Mean Absolute Error (MAE) as our metric (lower is better). We consider the following baselines: **Entity mean/median** calculates the mean/median label value for each entity in training data and predicts the mean/median value for the entity. **Global mean/median** calculates the global mean/median label value over the training data and predicts the same mean/median value across all entities. **Global zero** predicts zero for all entities. **LightGBM** learns a LightGBM (Ke *et al.*, 2017) regressor over the raw entity features to predict the numerical targets. Note that only information from the single entity table is used.

Experimental results. Results in Table 2 show our RDL implementation outperforms or matches baselines in all cases. A number of tasks, such as `driver-position` and `study-adverse`, have matching performance up to statistical significance, suggesting some room for improvement. We analyze this further in Appendix C, identifying one potential cause, suggesting an opportunity for improved performance for regression tasks.

4.3 Recommendation

Finally, we also introduce recommendation tasks on pairs of entities. The task is to predict a list of top K target entities given a source entity at a given seed time. The metric we use is Mean Average Precision (MAP) @K, where K is set per task (higher is better). We consider the following baselines: **Global popularity** computes the top K most popular target entities (by count) across the entire training table and predict the K globally popular target entities across all source entities. **Past visit** computes the top K most visited target entities for each source entity within the training table and predict those past-visited target entities for each entity. **LightGBM** learns a LightGBM (Ke *et al.*, 2017) classifier over the raw features of the source and target entities (concatenated) to predict the link. Additionally, global popularity and past visit ranks are also provided as inputs.

For recommendation, it is also important to ensure a certain density of links in the training data in order for there to be sufficient predictive signal. In Appendix A we report statistics on the average number of destination entities each source entity links to. For most tasks the density is ≥ 1, with the exception of `rel-stack` which is more sparse, but is included to test in extreme sparse settings.

Table 3: Recommendation results (MAP, higher is better) on RELBENCH. Best values are in bold. See Table 6 in Appendix B for standard deviations.

Dataset	Task	Split	Global Popularity	Past Visit	LightGBM	RDL (GraphSAGE)	RDL (ID-GNN)	Rel. Gain of RDL
rel-amazon	user-item-purchase	Val	0.31	0.07	0.18	**1.53**	0.13	397.55 %
		Test	0.24	0.06	0.16	**0.74**	0.10	204.74 %
	user-item-rate	Val	0.16	0.09	0.22	**1.42**	0.15	550.12 %
		Test	0.15	0.07	0.17	**0.87**	0.12	395.92 %
	user-item-review	Val	0.18	0.05	0.14	**1.03**	0.11	476.06 %
		Test	0.11	0.04	0.09	**0.47**	0.09	313.07 %
rel-avito	user-ad-visit	Val	0.01	3.66	0.17	0.09	**5.40**	47.37 %
		Test	0.00	1.95	0.06	0.02	**3.66**	87.09 %
rel-hm	user-item-purchase	Val	0.36	1.07	0.44	0.92	**2.64**	145.60 %
		Test	0.30	0.89	0.38	0.80	**2.81**	214.49 %
rel-stack	user-post-comment	Val	0.03	2.05	0.04	0.43	**15.17**	640.05 %
		Test	0.02	1.42	0.04	0.11	**12.72**	795.15 %
	post-post-related	Val	0.47	0.00	1.62	0.00	**7.76**	378.26 %
		Test	1.46	1.74	2.00	0.07	**10.83**	440.27 %
rel-trial	condition-sponsor-run	Val	2.63	8.58	4.88	3.12	**11.33**	32.05 %
		Test	2.52	8.42	4.82	2.89	**11.36**	34.89 %
	site-sponsor-run	Val	4.91	15.90	10.92	14.09	**17.43**	9.65 %
		Test	3.75	17.31	8.40	10.70	**19.00**	9.74 %
	Average	Val	1.01	3.50	2.07	2.51	**6.68**	297.41 %
		Test	0.95	3.55	1.79	1.85	**6.74**	277.26 %

Experimental results. Results are given in Table 3. We find that either the RDL implementation using GraphSAGE (Hamilton *et al.*, 2017), or ID-GNN (You *et al.*, 2021) as the GNN component performs best, often by a very significant margin. ID-GNN excels in cases were predictions are entity-specific (*i.e.*, Past Visit baseline outperforms Global Popularity), whilst the plain GNN excels in the reverse case. This reflects the inductive biases of each model, with GraphSAGE being able to learn structural features, and ID-GNN able to take into account the specific node ID.

5 Expert Data Scientist User Study

To rigorously test RDL, we conducted a human trial where a data scientist manually engineered features and used methods like LightGBM or XGBoost (Chen and Guestrin, 2016; Ke *et al.*, 2017). This represents the prior standard for building predictive models on relational databases (Heaton, 2016), providing a key comparison for RDL.

The study follows five workflow steps: **EDA:** Exploring the dataset to understand its characteristics, including feature columns and missing data. **Feature ideation:** Proposing entity-level features that may contain predictive signals. **Feature engineering:** Using SQL to compute and add features to the target table. **Tabular ML:** Running LightGBM or XGBoost on the table with engineered features and recording performance. **Post-hoc feature analysis (Optional):** Tools like SHAP and LIME explain feature contributions.

For example, in the `rel-hm` dataset, additional features such as *time since last purchase* are computed to predict customer churn. A detailed walkthrough of the data scientist's process is provided in Appendix C.

Limitations of Manual Feature Engineering. This process is labor-intensive, misses potential signals, and limits feature complexity. Every new task requires repeating these steps, adding hours of human labor and SQL code (Zheng and Casari, 2018). RDL models avoid these issues.

Data Scientist. We recruited a data scientist with a Stanford CS MSc, 4.0 GPA, and five years of experience building ML models, following the five steps outlined above.

User Study Protocol. The study protocol standardizes the time spent at each step: **EDA:** Capped at 4 hours to understand the dataset's schema and relationships. **Feature ideation:** Limited to 1 hour, with features proposed manually. **Feature engineering:** SQL queries are used to generate features, with no time limit for code writing. The time spent is recorded. **Tabular ML:** A standardized LightGBM script is used, with time recorded for preprocessing SQL query results. **Post-hoc analysis:** Conducted for sanity checks, taking just a few minutes (not included in total time).

Results. We compare RDL to the data scientist on three metrics: (i) predictive power, (ii) hours of human work, and (iii) lines of code. Marginal effort was measured, excluding reusable infrastructure. Figures 2, 3, and 4 show that RDL outperforms the data scientist in 11 of 15 tasks while reducing

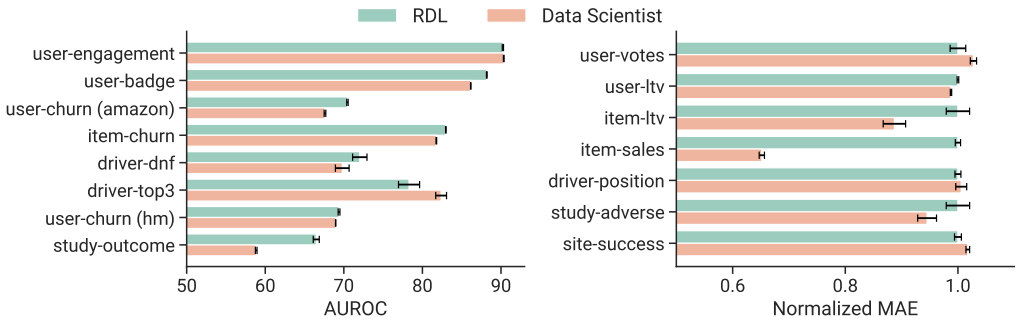


Figure 2: **RDL vs. Data Scientist.** RDL matches or outperforms the data scientist in 11 of 15 tasks. Left: AUROC for classification, right: MAE (normalized) for regression.

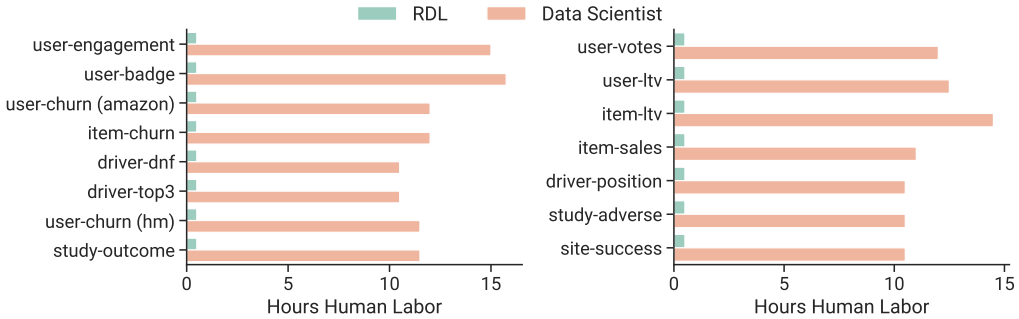


Figure 3: **RDL vs. Data Scientist.** RDL reduces the human work required to solve a task by 96% on average. Left: classification, right: regression.

hours worked by 96% and lines of code by 94%. On average, the data scientist spent 12.3 hours per task, while RDL took just 30 minutes. This demonstrates the potential of RDL to transform predictive tasks on relational databases, replacing manual feature engineering with end-to-end learnable models, a key insight from the last 15 years of AI research. RDL outperforms the data scientist in classification tasks but struggles more on regression. Improvements in output heads for regression could enhance RDL's performance. RDL reduces hours worked by 96% and lines of code by 94%. Much of RDL's code is reusable, while the data scientist must write task-specific code for each problem, highlighting RDL's efficiency advantage.

6 Conclusion

This work introduces RELBENCH, a benchmark to facilitate research on relational deep learning (Fey *et al.*, 2024). RELBENCH provides diverse and realistic relational databases and define practical predictive tasks that cover both entity-level prediction and entity link prediction. In addition, we provide the first open-source implementation of relational deep learning and validated its effectiveness over the common practice of manual feature engineering by an experienced data scientist. We hope RELBENCH will catalyze further research on relational deep learning to achieve highly-accurate prediction over complex multi-tabular datasets without manual feature engineering.

Acknowledgments and Disclosure of Funding

We thank Shirley Wu, Kaidi Cao, Rok Sosic, Yu He, Qian Huang, Bruno Ribeiro and Michi Yasunaga for discussions and for providing feedback on our manuscript. We also gratefully acknowledge the support of NSF under Nos. OAC-1835598 (CINES), CCF-1918940 (Expeditions), DMS-2327709 (IHBEM); Stanford Data Applications Initiative, Wu Tsai Neurosciences Institute, Stanford Institute for Human-Centered AI, Chan Zuckerberg Initiative, Amazon, Genentech, GSK, Hitachi, SAP, and UCB. The content is solely the responsibility of the authors and does not necessarily represent the official views of the funding entities.

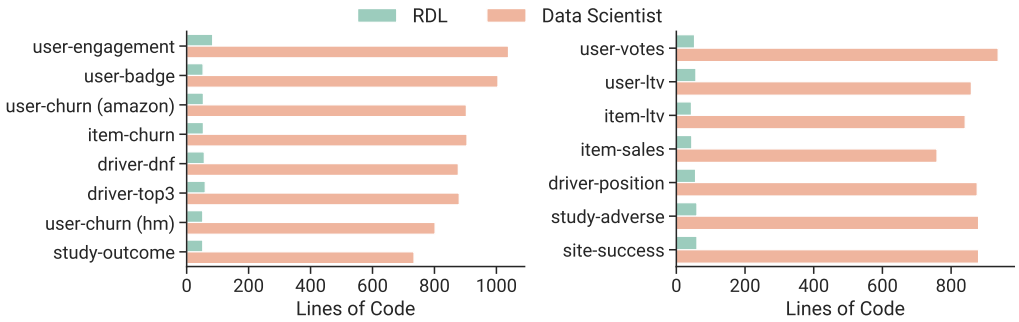


Figure 4: **RDL vs. Data Scientist.** RDL reduces new lines of code by 94%. Left: classification, right: regression.

References

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 *ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, 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.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *ICLR 2019 (RLGM Workshop)*, 2019.

Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Relational deep learning: Graph representation learning on relational databases. *ICML Position Paper*, 2024.

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 *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

James A Hanley and Barbara J McNeil. A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology*, 148(3):839–843, 1983.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

Jeff Heaton. An empirical analysis of feature engineering for predictive modeling. In *SoutheastCon 2016*, pages 1–6. IEEE, 2016.

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

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

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, 2017.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

Jianmo Ni, Jiacheng Li, and Julian McAuley. Justifying recommendations using distantly-labeled reviews and fine-grained aspects. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pages 188–197, 2019.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

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

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, *et al.* Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

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.

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.

Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, pages 10737–10745, 2021.

Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. " O'Reilly Media, Inc.", 2018.

A Additional Task Information

For reference, the following list documents all the predictive tasks in RELBENCH.

1. `rel-amazon`

   Node-level tasks:

   (a) `user-churn`: For each user, predict 1 if the customer does not review any product in the next 3 months, and 0 otherwise.

   (b) `user-ltv`: For each user, predict the $ value of the total number of products they buy and review in the next 3 months.

   (c) `item-churn`: For each product, predict 1 if the product does not receive any reviews in the next 3 months.

   (d) `item-ltv`: For each product, predict the $ value of the total number purchases and reviews it recieves in the next 3 months.

   Link-level tasks:

   (a) `user-item-purchase`: Predict the list of distinct items each customer will purchase in the next 3 months.

   (b) `user-item-rate`: Predict the list of distinct items each customer will purchase and give a 5 star review in the next 3 months.

   (c) `user-item-review`: Predict the list of distinct items each customer will purchase and give a detailed review in the next 3 months.

2. `rel-avito`

   Node-level tasks:

   (a) `user-visits`: Predict whether each customer will visit more than one Ad in the next 4 days.

   (b) `user-clicks`: Predict whether each customer will click on more than one Ads in the next 4 day.

   (c) `ad-ctr`: Assuming the Ad will be clicked in the next 4 days, predict the Click-Through-Rate (CTR) for each Ad.

   Link-level tasks:

   (a) `user-ad-visit`: Predict the list of ads a user will visit in the next 4 days.

3. `rel-f1`

   Node-level tasks:

   (a) `driver-position`: Predict the average finishing position of each driver all races in the next 2 months.

   (b) `driver-dnf`: For each driver predict the if they will DNF (did not finish) a race in the next 1 month.

   (c) `driver-top3`: For each driver predict if they will qualify in the top-3 for a race in the next 1 month.

4. `rel-hm`

   Node-level tasks:

   (a) `user-churn`: Predict the churn for a customer (no transactions) in the next week.

   (b) `item-sales`: Predict the total sales for an article (the sum of prices of the associated transactions) in the next week.

   Link-level tasks:

   (a) `user-item-purchase`: Predict the list of articles each customer will purchase in the next seven days.

5. `rel-stack`

   Node-level tasks:

   (a) `user-engagement`: For each user predict if a user will make any votes, posts, or comments in the next 3 months.

(b) `post-votes`: For each user post predict how many votes it will receive in the next 3 months

(c) `user-badge`: For each user predict if each user will receive in a new badge the next 3 months.

Link-level tasks:

(a) `user-post-comment`: Predict a list of existing posts that a user will comment in the next two years.

(b) `post-post-related`: Predict a list of existing posts that users will link a given post to in the next two years.

6. `rel-trial`

Node-level tasks:

(a) `study-outcome`: Predict if the trials in the next 1 year will achieve its primary outcome.

(b) `study-adverse`: Predict the number of affected patients with severe advsere events/death for the trial in the next 1 year.

(c) `site-success`: Predict the success rate of a trial site in the next 1 year.

Link-level tasks:

(a) `condition-sponsor-run`: Predict whether this condition will have which sponsors.

(b) `site-sponsor-run`: Predict whether this sponsor will have a trial in a facility.

7. `rel-event`

Node-level tasks:

(a) `user-attendance`: Predict how many events each user will respond yes or maybe in the next seven days.

(b) `user-repeat`: Predict whether a user will attend an event(by responding yes or maybe) in the next 7 days if they have already attended an event in the last 14 days.

(c) `user-ignore`: Predict whether a user will ignore more than 2 event invitations in the next 7 days.

B Experiment Details and Additional Results

B.1 Detailed Results

Tables 4, 5 and 6 show mean and standard deviations over 5 runs for the entity classification, entity regression and link prediction results respectively.

B.2 Hyperparameter Choices

All our RDL experiments were run based on a single set of default task-specific hyperparameters, *i.e.* we did not perform exhaustive hyperparamter tuning, *cf.* Table 7. This verifies the stability and robustness of RDL solutions, even against expert data scientist baselines. Specifically, all task types use a shared GNN configuration (a two-layer GNN with a hidden feature size of 128 and "sum" aggregation) and sample subgraphs identically (disjoint subgraphs of 512 seed entities with a maximum of 128 neighbors for each foreign key). Across task types, we only vary the learning rate and the maximum number of epochs to train for.

Notably, we found that our default set of hyperparameters heavily underperformed on the node-level tasks on the `rel-trial` dataset. On this dataset, we used a learning rate of 0.0001, a "mean" neighborhood aggregation scheme, 64 sampled neighbors, and trained for a maximum of 20 epochs. For the ID-GNN link-prediction experiments on `rel-trial`, it was necessary to use a four-layer deep GNN in order to ensure that destination nodes are part of source node-centric subgraphs.

B.3 Ablations

We also report additional results ablating parts of our relational deep learning implementation. All experiments are designed to be data-centric, aiming to validate basic properties of the chosen datasets

Table 4: Entity classification results (AUROC mean$_{\pm \text{std}}$ over 5 runs, higher is better) on RELBENCH. Best values are in bold along with those not statistically different from it.

Dataset	Task	Split	LightGBM	RDL
rel-amazon	user-churn	Val	$52.05_{\pm 0.06}$	$\mathbf{70.45}_{\pm 0.06}$
		Test	$52.22_{\pm 0.06}$	$\mathbf{70.42}_{\pm 0.05}$
	item-churn	Val	$62.39_{\pm 0.20}$	$\mathbf{82.39}_{\pm 0.02}$
		Test	$62.54_{\pm 0.18}$	$\mathbf{82.81}_{\pm 0.03}$
rel-avito	user-visits	Val	$53.31_{\pm 0.09}$	$\mathbf{69.65}_{\pm 0.04}$
		Test	$53.05_{\pm 0.32}$	$\mathbf{66.20}_{\pm 0.10}$
	user-clicks	Val	$55.63_{\pm 0.31}$	$\mathbf{64.73}_{\pm 0.32}$
		Test	$53.60_{\pm 0.59}$	$\mathbf{65.90}_{\pm 1.95}$
rel-event	user-repeat	Val	$\mathbf{67.76}_{\pm 0.97}$	$\mathbf{71.25}_{\pm 2.53}$
		Test	$68.04_{\pm 1.82}$	$\mathbf{76.89}_{\pm 1.59}$
	user-ignore	Val	$87.96_{\pm 0.28}$	$\mathbf{91.70}_{\pm 0.33}$
		Test	$79.93_{\pm 0.49}$	$\mathbf{81.62}_{\pm 1.11}$
rel-f1	driver-dnf	Val	$68.42_{\pm 1.14}$	$\mathbf{71.36}_{\pm 1.54}$
		Test	$\mathbf{68.56}_{\pm 3.89}$	$\mathbf{72.62}_{\pm 0.27}$
	driver-top3	Val	$67.76_{\pm 2.75}$	$\mathbf{77.64}_{\pm 3.16}$
		Test	$\mathbf{73.92}_{\pm 5.75}$	$\mathbf{75.54}_{\pm 0.63}$
rel-hm	user-churn	Val	$56.05_{\pm 0.05}$	$\mathbf{70.42}_{\pm 0.09}$
		Test	$55.21_{\pm 0.12}$	$\mathbf{69.88}_{\pm 0.21}$
rel-stack	user-engagement	Val	$65.12_{\pm 0.25}$	$\mathbf{90.21}_{\pm 0.07}$
		Test	$63.39_{\pm 0.26}$	$\mathbf{90.59}_{\pm 0.09}$
	user-badge	Val	$65.39_{\pm 0.05}$	$\mathbf{89.86}_{\pm 0.08}$
		Test	$63.43_{\pm 0.12}$	$\mathbf{88.86}_{\pm 0.08}$
rel-trial	study-outcome	Val	$\mathbf{68.30}_{\pm 0.53}$	$\mathbf{68.18}_{\pm 0.49}$
		Test	$\mathbf{70.09}_{\pm 1.41}$	$\mathbf{68.60}_{\pm 1.01}$

Table 5: Entity regression results (MAE mean$_{\pm \text{std}}$ over 5 runs, lower is better) on RELBENCH. Best values are in bold along with those not statistically different from it.

Dataset	Task	Split	Global Zero	Global Mean	Global Median	Entity Mean	Entity Median	LightGBM	RDL
rel-amazon	user-ltv	Val	14.141	20.740	14.141	17.685	15.978	$14.141_{\pm 0.000}$	$\mathbf{12.132}_{\pm 0.007}$
		Test	16.783	22.121	16.783	19.055	17.423	$16.783_{\pm 0.000}$	$\mathbf{14.313}_{\pm 0.013}$
	item-ltv	Val	72.096	78.110	59.471	80.466	68.922	$55.741_{\pm 0.049}$	$\mathbf{45.140}_{\pm 0.068}$
		Test	77.126	81.852	64.234	78.423	66.436	$60.569_{\pm 0.047}$	$\mathbf{50.053}_{\pm 0.163}$
rel-avito	ad-ctr	Val	0.048	0.048	0.040	0.044	0.044	$0.037_{\pm 0.000}$	$\mathbf{0.037}_{\pm 0.000}$
		Test	0.052	0.051	0.043	0.046	0.046	$\mathbf{0.041}_{\pm 0.000}$	$\mathbf{0.041}_{\pm 0.001}$
rel-event	user-attendance	Val	0.262	0.457	0.262	0.296	0.268	$0.262_{\pm 0.000}$	$\mathbf{0.255}_{\pm 0.007}$
		Test	**0.264**	0.470	**0.264**	0.304	0.269	$\mathbf{0.264}_{\pm 0.000}$	$\mathbf{0.258}_{\pm 0.006}$
rel-f1	driver-position	Val	11.083	4.334	4.136	7.181	7.114	$3.450_{\pm 0.030}$	$\mathbf{3.193}_{\pm 0.024}$
		Test	11.926	4.513	4.399	8.501	8.519	$\mathbf{4.170}_{\pm 0.137}$	$\mathbf{4.022}_{\pm 0.119}$
rel-hm	item-sales	Val	0.086	0.142	0.086	0.117	0.086	$0.086_{\pm 0.000}$	$\mathbf{0.065}_{\pm 0.000}$
		Test	0.076	0.134	0.076	0.111	0.078	$0.076_{\pm 0.000}$	$\mathbf{0.056}_{\pm 0.000}$
rel-stack	post-votes	Val	0.062	0.146	0.062	0.102	0.064	$0.062_{\pm 0.000}$	$\mathbf{0.059}_{\pm 0.000}$
		Test	0.068	0.149	0.068	0.106	0.069	$0.068_{\pm 0.000}$	$\mathbf{0.065}_{\pm 0.000}$
rel-trial	study-adverse	Val	57.083	75.008	56.786	57.083	57.083	$\mathbf{45.774}_{\pm 1.191}$	$\mathbf{46.290}_{\pm 0.304}$
		Test	57.930	73.781	57.533	57.930	57.930	$\mathbf{44.011}_{\pm 0.998}$	$\mathbf{44.473}_{\pm 0.209}$
	site-success	Val	0.475	0.462	0.475	0.447	0.450	$0.417_{\pm 0.003}$	$\mathbf{0.401}_{\pm 0.009}$
		Test	0.462	0.468	0.462	0.448	0.441	$0.425_{\pm 0.003}$	$\mathbf{0.400}_{\pm 0.020}$

Table 6: Link prediction results (MAP mean$_{\pm \text{std}}$ over 5 runs, higher is better) on RELBENCH. Best values are in bold along with those not statistically different from it.

Dataset	Task	Split	Global Popularity	Past Visit	LightGBM	RDL (GraphSAGE)	RDL (ID-GNN)
rel-amazon	user-item-purchase	Val	0.31	0.07	$0.18_{\pm 0.07}$	$\mathbf{1.53}_{\pm 0.05}$	$0.13_{\pm 0.00}$
		Test	0.24	0.06	$0.16_{\pm 0.05}$	$\mathbf{0.74}_{\pm 0.08}$	$0.10_{\pm 0.00}$
	user-item-rate	Val	0.16	0.09	$0.22_{\pm 0.02}$	$\mathbf{1.42}_{\pm 0.06}$	$0.15_{\pm 0.00}$
		Test	0.15	0.07	$0.17_{\pm 0.01}$	$\mathbf{0.87}_{\pm 0.05}$	$0.12_{\pm 0.00}$
	user-item-review	Val	0.18	0.05	$0.14_{\pm 0.03}$	$\mathbf{1.03}_{\pm 0.03}$	$0.11_{\pm 0.00}$
		Test	0.11	0.04	$0.09_{\pm 0.01}$	$\mathbf{0.47}_{\pm 0.05}$	$0.09_{\pm 0.00}$
rel-avito	user-ad-visit	Val	0.01	3.66	$0.17_{\pm 0.01}$	$0.09_{\pm 0.01}$	$\mathbf{5.40}_{\pm 0.02}$
		Test	0.00	1.95	$0.06_{\pm 0.01}$	$0.02_{\pm 0.00}$	$\mathbf{3.66}_{\pm 0.02}$
rel-hm	user-item-purchase	Val	0.36	1.07	$0.44_{\pm 0.03}$	$0.92_{\pm 0.04}$	$\mathbf{2.64}_{\pm 0.00}$
		Test	0.30	0.89	$0.38_{\pm 0.02}$	$0.80_{\pm 0.03}$	$\mathbf{2.81}_{\pm 0.01}$
rel-stack	user-post-comment	Val	0.03	2.05	$0.04_{\pm 0.02}$	$0.43_{\pm 0.08}$	$\mathbf{15.17}_{\pm 0.15}$
		Test	0.02	1.42	$0.04_{\pm 0.03}$	$0.11_{\pm 0.05}$	$\mathbf{12.72}_{\pm 0.22}$
	post-post-related	Val	0.47	0.00	$1.62_{\pm 0.36}$	$0.00_{\pm 0.01}$	$\mathbf{7.76}_{\pm 0.20}$
		Test	1.46	1.74	$2.00_{\pm 0.43}$	$0.07_{\pm 0.08}$	$\mathbf{10.83}_{\pm 0.22}$
rel-trial	condition-sponsor-run	Val	2.63	8.58	$4.88_{\pm 0.13}$	$3.12_{\pm 0.24}$	$\mathbf{11.33}_{\pm 0.04}$
		Test	2.52	8.42	$4.82_{\pm 0.20}$	$2.89_{\pm 0.39}$	$\mathbf{11.36}_{\pm 0.08}$
	site-sponsor-run	Val	4.91	15.90	$10.92_{\pm 0.67}$	$14.09_{\pm 0.77}$	$\mathbf{17.43}_{\pm 0.07}$
		Test	3.75	17.31	$8.40_{\pm 0.70}$	$10.70_{\pm 1.10}$	$\mathbf{19.00}_{\pm 0.12}$

Table 7: Task-specific RDL default hyperparameters.

Hyperparameter	Task type		
	Node classification	Node regression	Link prediction
Learning rate	0.005	0.005	0.001
Maximum epochs	10	10	20
Batch size	512	512	512
Hidden feature size	128	128	128
Aggregation	summation	summation	summation
Number of layers	2	2	2
Number of neighbors	128	128	128
Temporal sampling strategy	uniform	uniform	uniform

and tasks. Examples include confirming that the graph structure, node features, and temporal-awareness all play important roles in achieving optimal performance, which also underscores the unique challenges our RELBENCH dataset and tasks present.

Graph structure. We first investigate the role of the graph structure we adopt for GNNs on RELBENCH. Specifically, we compare the following two approaches of constructing the edges: **1.** *Primary-foreign key (pkey-fkey)*, where the entities from two tables that share the same primary key and foreign key are connected through an edge; **2.** *Randomly permuted*, where we apply a random permutation on the destination nodes in the primary-foreign key graph for each type of the edge while keeping the source nodes untouched. From Fig. 5 we observe that with random permutation on the primary-foreign key edges the performance of the GNN becomes much worse, verifying the critical role of carefully constructing the graph structure through, *e.g.*, primary-foreign key as proposed in Fey *et al.* (2024).

Node features and text embeddings. Here we study the effect of node features used in RELBENCH. In the experiments depicted in Fig. 6, we compare GNN (w/ node feature) with its variant where the node features are all masked by zeros (*i.e.*, w/o node feature). We find that utilizing rich node features incorporated in our RELBENCH dataset is crucial for GNN. Moreover, we also investigate, in particular, the approach to encode texts in the data that constitutes part of the node features. In Fig. 7, we compare GloVe text embedding (Pennington *et al.*, 2014) and BERT text embedding (Devlin *et al.*, 2018) with w/o text embedding, where the text embeddings are masked by zeros. We observe that encoding the rich texts in RELBENCH with GloVe or BERT embedding consistently yields better performance compared with using no text features. We also find that BERT embedding is usually better than GloVe embedding especially for node classification tasks, which suggests that enhancing the quality of text embedding will potentially help achieve better performance.

Temporal awareness. We also investigate the importance of injecting temporal awareness into the GNN by ablating on the time embedding. To be specific, in the implementation we add a relative

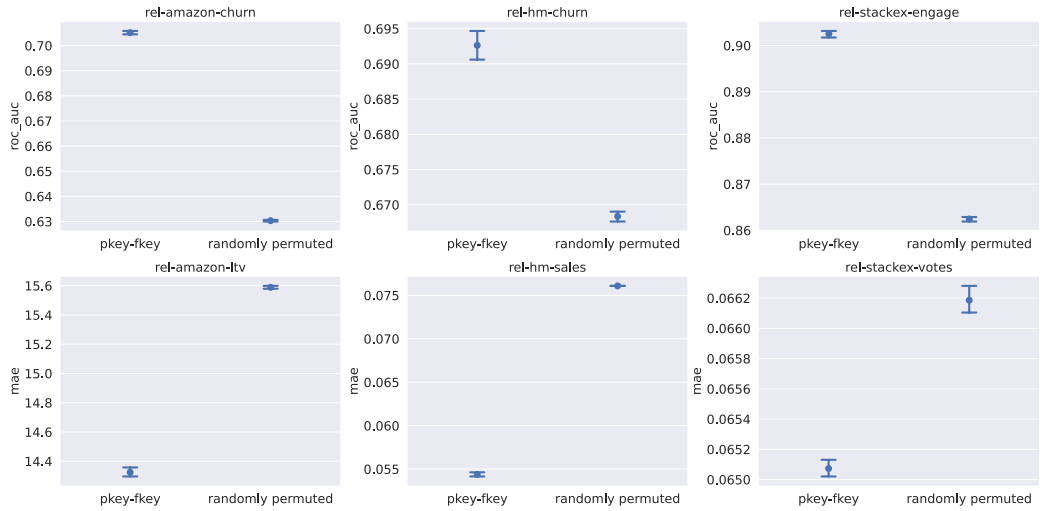


Figure 5: Investigation on the role of leveraging primary-foreign key (pkey-fkey) edges for the GNN. At the top row are three node classification tasks with metric AUROC (higher is better) while at the bottom are three node regression tasks with metric MAE (lower is better), evaluated on the test set. We find that our proposal of using pkey-fkey edges for message passing is vital for GNN to achieve desirable performance on RELBENCH. Error bars correspond to 95% confidence interval.

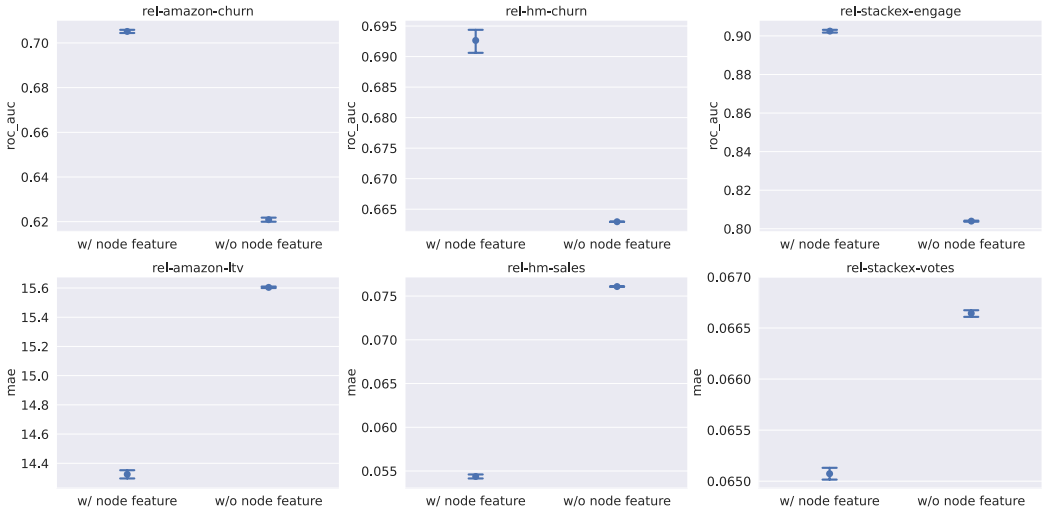


Figure 6: Investigation on the role of node features. At the top row are three node classification tasks with metric AUROC (higher is better) while at the bottom are three node regression tasks with metric MAE (lower is better), evaluated on the test set. We observe that leveraging node features is important for GNN. Error bars correspond to 95% confidence interval.

time embedding when deriving the node features using the relative time span between the timestamp of the entity and the querying seed time. Results are exhibited in Fig. 8. We discover that adding the time embedding significantly enhance the performance across a diverse range of tasks, demonstrating the efficacy and importance of building up the temporal awareness into the model.

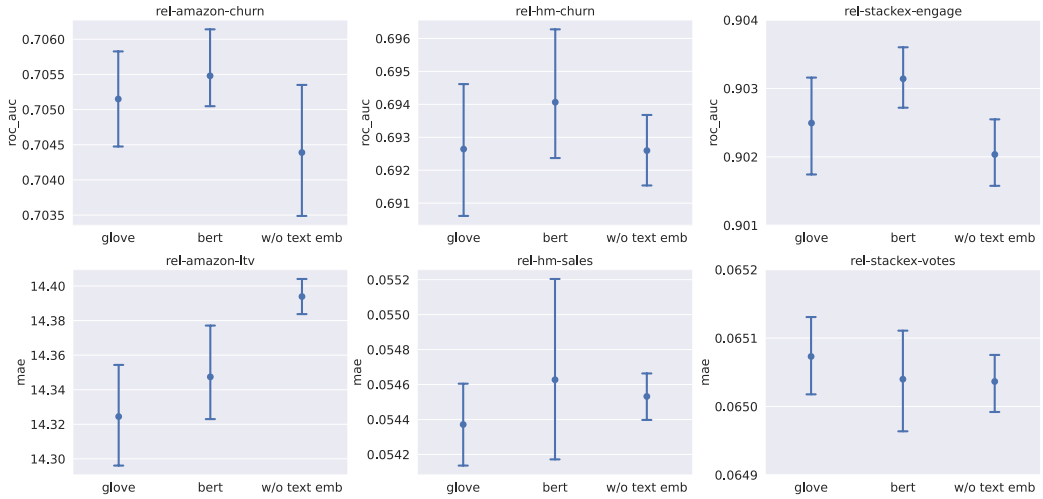


Figure 7: Investigation on the role of text embedding. At the top row are three node classification tasks with metric AUROC (higher is better) while at the bottom are three node regression tasks with metric MAE (lower is better), evaluated on the test set. We observe that adding text embedding using GloVe (Pennington *et al.*, 2014) or BERT (Devlin *et al.*, 2018) generally helps improve the performance. Error bars correspond to 95% confidence interval.

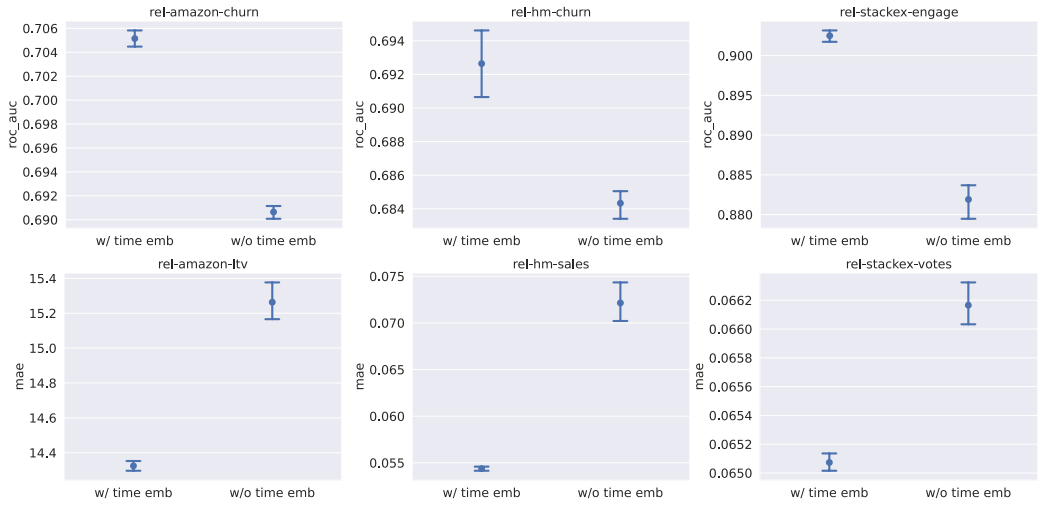


Figure 8: Investigation on the role of time embedding. At the top row are three node classification tasks with metric AUROC (higher is better) while at the bottom are three node regression tasks with metric MAE (lower is better), evaluated on the test set. We find that adding time embedding to the GNN consistently boosts the performance. Error bars correspond to 95% confidence interval.

C User Study Additional Details

C.1 Data Scientist Example Workflow

In this section we provide a detailed description of the data scientist workflow for the `user-churn` task of the `rel-hm` dataset. The purpose of this is to exemplify the efforts undertaken by the data scientist to solve RELBENCH tasks. For data scientist solutions to all tasks, see https://github.com/snap-stanford/relbench-user-study.

Recall that the main data science workflow steps are:

1. Exploratory data analysis (EDA).

2. Feature ideation.
3. Feature enginnering.
4. Tabular ML.
5. Post-hoc analysis of feature importance (optional).

C.1.1 Exploratory Data Analysis

During the exploratory data analysis (EDA) the data scientist familiarizes themselves with a new dataset. It is typically carried out in a Jupyter notebook, where the data scientist first loads the dataset or establishes a connection to it and then systematically explores it. The data scientist may:

- Visualize the database schema, looking at the fields of different tables and the relationships between them.
- Closely analyze the label sets:
  - Look at the relative sizes and temporal split of the training, validation and test subsets.
  - Look at label statistics such as the mean, the standard deviation and various quantiles.
  - For classification tasks, understand class (im)balance: how much bigger is the modal class than the rest? For example, in the `user-churn` task roughly 82% of the samples have label 1, so there is a good amount of imbalance but not enough to strictly require up-sampling techniques.
  - For regression tasks, understand the label distribution: are the labels concentrated around a typical value or do they follow a power law wherein the labels span several orders of magnitude? In extreme cases, this exploration will point to a need for specialized handing of the label space for model training.
- Plot distributions and aggregations of interesting columns/fields. For example, in Figure 9 we can see three such plots. From left to right:
  - The first plot shows the distribution of age among customers. We see two distinct peaks one in the mid-twenties and another in the mid-fifties, suggesting different customer "archetypes", which may have different spending patterns.
  - The second plot shows the number of sales per month over a two year period. We can see some seasonality with summer months being particularly good for overall sales. This suggests date related features could be useful.
  - The third plot shows a *Lorenz curve* of sales per article, showcasing the canonical *Pareto Principle*: 20% of the articles account for 80% of the sales.
- Run custom queries to look at interesting quantities and/or relationships between different columns. For instance, in the EDA for `rel-hm`, an interesting quantity to look at is the variability in item prices across the year. This reveals that most of the variability is downward, representing temporary discounts.
- Investigate outliers or odd-looking patterns in the data. These usually will have some real-world explanation that may inform how the data scientist chooses to pre-process the data and construct features.

In all, this process takes in the order of a few hours (3-4 for most datasets in the user study).

C.1.2 Feature Ideation

Having explored the dataset in the EDA, the data scientist will then brainstorm features that, to their judgement, will provide valuable signal to a model for a specific learning task. In the case of the `user-churn` task, a rather simple feature would be the customer's age, which is a field directly available in one of the tables. A slightly more complex feature would be the total amount spent by the customer so far. Finally, an example of a fairly complex feature is the average monthly sales volume of items purchased by the customer in the past week. A high value for this feature may indicate that the customer has been shopping trendy items lately, whereas a low value for this feature may indicate that the customer has been interested in more arcane or specific items.

In practice, the ideation phase consists of writing down all of these feature ideas in a file or a piece of paper. It is the quickest part of the whole process and in this user study took between 30 minutes and one hour.

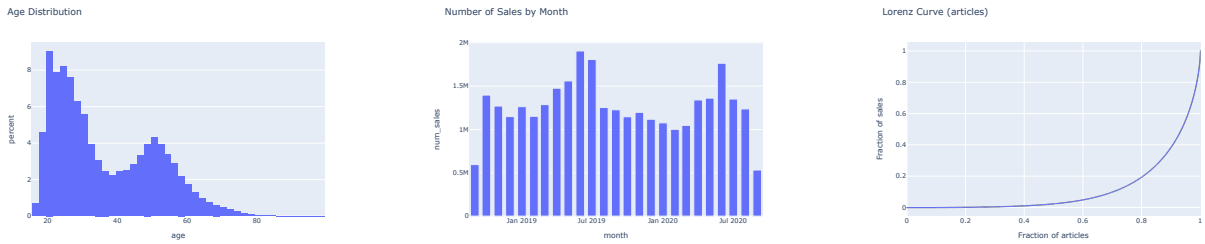


Figure 9: **EDA Plots.** Each plot explores different characteristics of the dataset. Understanding the data and identifying relationships between different quantities is an essential prerequisite to meaningful feature engineering.

C.1.3 Feature Engineering

With a list of features in hand, the data scientist then proceeds to actually write code to generate all the features for each sample in the the train, validation and test subsets. In this user study, this was carried out using DuckDB SQL[2] with some Jinja templating[3] for convenience.

Revisiting the example features from the previous section, the conceptual complexity of the features closely tracks with the technical complexity of implementing them. For customer age all that is required is a simple *join*. The total amount spent by the customer, can be calculated using a *group by* clause and a couple of *join*'s. Lastly, calculating the average monthly sales volume of items purchased by the customer in the past week requires multiple *group by*'s, *join*'s, and *window functions* distributed across multiple *common table expressions* (CTEs).

A key consideration during feature engineering is the prevention of *leakage*. The data scientist must ensure that none of the features accidentally include information from after the sample timestamp. This is especially true for complex features like the third example above, where special care must be taken to ensure that each *join* has the appropriate filters to comply with the sample timestamp.

For some tasks, *e.g.*, `study-outcome`, the initial features *did* leak information from the validation set into the training set. Thanks to the RELBENCH testing setup, leaking test data into the training data is hard to do by accident, since test data is hidden. Leaking information from validation to train (but not test to train) led to extremely high validation performance and very low test performance (test was significantly lower than LightGBM with no feature engineering). The large discrepancy between validation and test performances alerted the data scientist to the mistake, and the features were eventually fixed. This example illustrates another complexity that feature engineering introduces, with special care needed to ensure leakage does not happen.

Other considerations that the data scientist must keep in mind during development and implementation of the features are parsing issues, runtime constrains and memory load. For example, during the user study we identified a parsing issue arising from special characters in user posts/comments in the `rel-stack` dataset. The *backslash* character, widely used LaTeXcan trip up certain text parsers if not handled with care. Furthermore, runtime and memory constraints are important to keep in mind when working with larger datasets and computing features that require nested *join*'s and aggregations. During the user study, there were some cases where we had to refactor SQL queries to make them more efficient, increasing the overall implementation time. For some tasks we had to implement sub-sampling of the training set to reduce the burden on compute resources.

Finally, once the features have been generated for each data subset, the data scientist will usually inspect the generated features looking for anomalies (e.g. an unusual prevalence of *NULL* values). In this user study we also implemented some automated sanity checks to validate the generated features beyond manual inspection.

[2] See https://duckdb.org/.
[3] See https://jinja.palletsprojects.com/en/3.1.x/intro/.

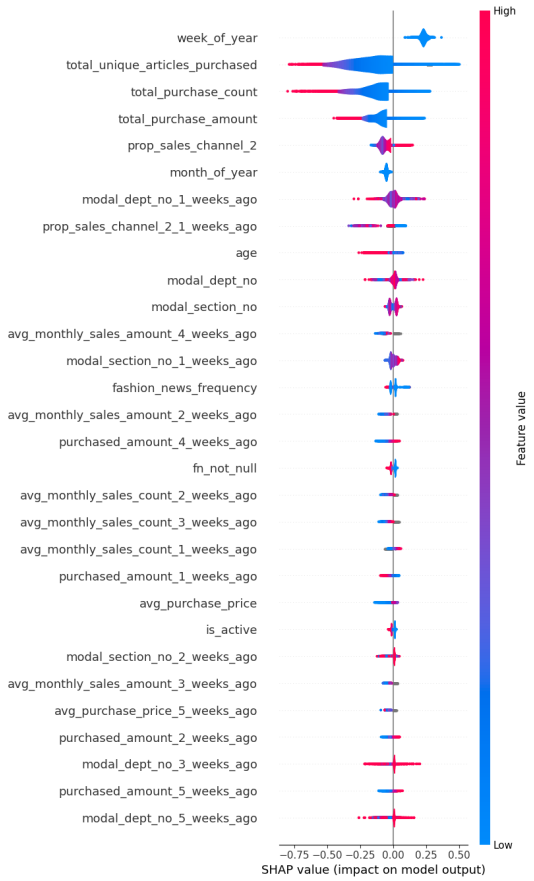


Figure 10: **Feature Importances.** SHAP values of top 30 features ranked by importance. Note: *week_of_year* feature shows little variability because the validation set is temporally concentrated in a few weeks.

C.1.4 Tabular Machine Learning

The output of the Feature Engineering phase is a DuckDB table with engineered features for each data subset. There is some non-trivial amount of work required to go from those tables to the numerical arrays used for training by most Tabular ML models (LightGBM in this case). This is implemented in a Python script that loads the data, transforms it into arrays and carries out hyperparameter tuning. In this user study we ran 5 hyperparameter optimization runs, with 10 trials each, reporting the mean and standard deviation over the 5 runs. For the `user-churn` task this took one to two hours.

C.1.5 Post-hoc Analysis

The last step in the process is to look at a trained model and analyze its performance and feature importance. To this end we used SHAP values (Lundberg and Lee, 2017) and the corresponding python package[4]. Figure 10 shows the top 30 most important features in the `user-churn` task. The individual *violin plots* show the distribution of SHAP values for a subset of the validation set, the color indicates the value of the feature. For the `user-churn` task, the most predictive features were primarily (1) all-time statistics of user behavior pattern, and (2) temporal information that allows the model to be aware of seasonality.

C.2 Regression Output Head Analysis

By default, our RDL implementation uses a simple linear output head on top of the GNN embeddings. However we found that on regression tasks this sometimes led to lower than desirable performance.

[4]See `https://shap.readthedocs.io/en/latest/`.

Table 8: Entity regression results (MAE, lower is better) on selected RELBENCH datasets. Training a LightGBM model on features extracted by a trained GNN leads to performance lift. This is evidence that the linear layer output head of the base GNN is suboptimal.

Dataset	Task	Split	GNN	GNN+LightGBM
rel-f1	driver-position	Test	$4.173_{\pm 0.178}$	$\mathbf{4.05}_{\pm 0.09}$
rel-stack	post-votes	Test	$0.065_{\pm 0.00}$	$\mathbf{0.062}_{\pm 0.00}$
rel-amazon	item-ltv	Test	$14.31_{\pm 0.028}$	$\mathbf{14.10}_{\pm 0.02}$

Table 9: Entity classification results (AUROC, higher is better, numbers bolded if withing standard deviation of best result) on selected RELBENCH tasks. Training a LightGBM model on features extracted by a trained GNN does not lead to performance lift, and can even hurt performance slightly. This is evidence that output head limitations hold for regression tasks only. Note, study-outcome uses default GNN parameters for simplicity, differing form the performance reported in the main paper.

Dataset	Task	Split	GNN	GNN+LightGBM
rel-f1	driver-dnf	Test	$\mathbf{72.3}_{\pm 1.67}$	$\mathbf{71.8}_{\pm 1.30}$
rel-trial	study-outcome	Test	$\mathbf{68.8}_{\pm 1.10}$	$\mathbf{68.2}_{\pm 0.44}$
rel-stack	user-badge	Test	$88.3_{\pm 0.04}$	$\mathbf{88.4}_{\pm 0.04}$

We found that performance on many regression tasks could be improved by modifying this output head. Instead of a linear layer, we took the output from the GNN, and fed these embeddings into a LightGBM model, which is trained in a second separate training phase from the GNN model.

The resulting model still uses an end-to-end learned GNN for cross-table feature engineering, showing that the GNN is learning useful features. Instead we attribute the weaker performance to the linear output head. We believe that further attention to the regression output head is an interesting direction for further study, with the goal of designing an output head that is performant and can be trained jointly with the GNN (unlike our LightGMB modification).

We run three experiments to study this phenomena, and attempt to isolate the output head as a problematic component for regression tasks.

1. LightGBM trained on GNN-learned entity-level features on regression tasks. We find that this model performs better than the original GNN, suggesting that the linear output head of the GNN is suboptimal.
2. LightGBM trained on GNN-learned entity-level features on classification tasks. We find no performance improvement, and even some degradation, compared to the original GNN model, suggesting that the observed performance boost of (1) comes not from an overall better architecture but from the correction of an innate shortcoming of the linear output head vis-a-vis regression tasks. In other words, using a LightGBM on top of the GNN is only helpful insofar as it provides a more flexible output head for regression tasks.
3. Evaluate GNN performance after converting regression tasks to binary classification tasks with label $y = \mathbf{1}\{y_{\text{regression}} > 0\}$. We find that the performance gap between the data scientist models and the GNN narrow. This suggests that the GNN can learn the relevant predictive signal, but performance is affected by how the task is formulated (classification vs regression).

See Tables 8, 9, 10 for the results of each of these experiments.

In Figure 2, for regression tasks we report the RDL results using GNN learned features with LightGBM output head. In Table 2 we report result for the basic GNN in order to avoid creating confusion for other researchers when comparing different GNN methods. We believe that Tables 8, 9, 10 provide clear evidence that there is an opportunity for improvements and simplifications, which we leave to future work.

D Dataset Origins and Licenes

This section details the sources for all data used in RELBENCH. In all cases, the data providers consent for their data to be used freely for non-commercial and research purposes. The only

Table 10: Entity classification results (AUROC, higher is better) on selected RELBENCH regression tasks, converted into classification tasks with binary label $y = \mathbf{1}\{y_{\text{regression}} > 0\}$. Training a LightGBM model on features extracted by a trained GNN leads to performance lift. This is evidence that the linear layer output head of the base GNN is suboptimal.

Dataset	Task	Split	GNN	Data Scientist
rel-f1	driver-position	Test	$81.96_{\pm 1.18}$	$\mathbf{86.63}_{\pm 0.40}$
rel-stack	post-votes	Test	$\mathbf{80.5}_{\pm 0.18}$	$78.3_{\pm 0.05}$
rel-amazon	item-ltv	Test	$\mathbf{70.61}_{\pm 0.06}$	$70.29_{\pm 0.06}$

database with potentially personally identifiable information is rel-stack, which draws from the Stack Exchange site, which sometimes has individuals' names as their username. This information shared with consent, as all users must agree to the Stack Exchange privacy policy, see: https://stackoverflow.com/legal/privacy-policy.

rel-amazon. Data obtained from the Amazon Review Data Dump from Ni *et al.* (2019). See the website: https://cseweb.ucsd.edu/~jmcauley/datasets/amazon_v2/. Data license is not specified.

rel-avito. Data is obtained from Kaggle https://www.kaggle.com/competitions/avito-context-ad-clicks. All RELBENCH users must download data from Kaggle themselves, a part of which is accepting the data usage terms. These terms include use only for non-commercial and academic purposes. Note that after data download, we further downsample the avito dataset by randomly selecting approximately 100,000 data point from user table and sample all other tables that have connections to the sampled users.

rel-stack. Data was obtained from The Internet Archive, whose stated mission is to provide "universal access to all knowledge. We downloaded our data from https://archive.org/download/stackexchange in Novermber 2023. Data license is not specified.

rel-f1. Data was sourced from the Ergast API (https://ergast.com/mrd/) in February 2024. The Ergast Developer API is an experimental web service which provides a historical record of motor racing data for non-commercial purposes. As far as we are able to determine the data is public and license is not specified.

rel-trial. Data was downloaded from the ClinicalTrials.gov website in January 2024. This data is provided by the NIH, an official branch of the US Government. The terms of use state that data are available to all requesters, both within and outside the United States, at no charge. Our rel-trial database is a snapshot from January 2024, and will not be updated with newer trials results.

rel-hm. Data is obtained from Kaggle https://www.kaggle.com/competitions/h-and-m-personalized-fashion-recommendations. All RELBENCH users must download data from Kaggle themselves, a part of which is accepting the data usage terms. These terms include use only for non-commercial and academic purposes.

rel-event. The dataset employed in this research was initially released on Kaggle for the Event Recommendation Engine Challenge, which can be accessed at https://www.kaggle.com/c/event-recommendation-engine-challenge/data. We have obtained explicit consent from the creators of this dataset to use it within RELBENCH. We extend our sincere gratitude to Allan Carroll for his support and generosity in sharing the data with the academic community.

E Additional Training Table Statistics

We report additional training table statistics for all tasks, separated into entity classification (*cf.* Table 11), entity regression (*cf.* Table 12), and link prediction (*cf.* Table 13).

Table 11: RELBENCH entity classification training table target statistics.

Dataset	Task	Split	Positives	Negatives
rel-amazon	user-churn	Train	2,956,658 (62.47%)	1,775,897 (37.53%)
		Val	263,098 (64.2%)	146,694 (35.8%)
		Test	213,400 (60.64%)	138,485 (39.36%)
	item-churn	Train	1,113,863 (43.52%)	1,445,401 (56.48%)
		Val	73,242 (41.22%)	104,447 (58.78%)
		Test	61,647 (36.95%)	105,195 (63.05%)
rel-f1	driver-dnf	Train	1,365 (11.96%)	10,046 (88.04%)
		Val	125 (22.08%)	441 (77.92%)
		Test	207 (29.49%)	495 (70.51%)
	driver-top3	Train	231 (17.07%)	1,122 (82.93%)
		Val	119 (20.24%)	469 (79.76%)
		Test	128 (17.63%)	598 (82.37%)
rel-hm	user-churn	Train	3,170,367 (81.89%)	701,043 (18.11%)
		Val	62,225 (81.28%)	14,331 (18.72%)
		Test	61,609 (82.61%)	12,966 (17.39%)
rel-stack	user-engagement	Train	68,020 (5.0%)	1,292,830 (95.0%)
		Val	2,411 (2.81%)	83,427 (97.19%)
		Test	2,411 (2.74%)	85,726 (97.26%)
	user-badge	Train	163,048 (4.81%)	3,223,228 (95.19%)
		Val	7,301 (2.95%)	240,097 (97.05%)
		Test	6,735 (2.64%)	248,625 (97.36%)
rel-trial	study-outcome	Train	7,647 (63.76%)	4,347 (36.24%)
		Val	561 (58.44%)	399 (41.56%)
		Test	483 (58.55%)	342 (41.45%)
rel-event	user-repeat	Train	1,882 (48.98%)	1,960 (51.02%)
		Val	130 (48.51%)	138 (51.49%)
		Test	110 (44.72%)	136 (55.28%)
	user-ignore	Train	3,247 (16.88%)	15,992 (83.12%)
		Val	441 (10.54%)	3,744 (89.46%)
		Test	450 (11.40%)	3,499 (88.60%)
rel-avito	user-clicks	Train	2,302 (3.87%)	57,152 (96.13%)
		Val	745 (3.52%)	20,438 (96.48%)
		Test	740 (1.54%)	47,256 (98.46%)
	user-visits	Train	78,467 (90.59%)	8,152 (9.41%)
		Val	27,086 (90.35%)	2,893 (9.65%)
		Test	30,731 (85.06%)	5,398 (14.94%)

Table 12: RELBENCH entity regression training table target statistics.

Dataset	Task	Split	Minimum	Median	Mean	Maximum
rel-amazon	user-ltv	Train	0.0	0.0	16.93	9,511.46
		Val	0.0	0.0	14.14	7,259.91
		Test	0.0	0.0	16.78	10,329.86
		Total	0.0	0.0	16.71	10,329.86
	item-ltv	Train	0.0	20.78	67.57	198,419.8
		Val	0.0	22.44	72.10	75,901.55
		Test	0.0	23.72	77.13	206,663.58
		Total	0.0	20.97	68.38	206,663.58
rel-f1	driver-position	Train	1.0	13.33	13.90	39.0
		Val	1.0	11.4	11.08	22.0
		Test	1.0	12.18	11.93	24.0
		Total	1.0	13.0	13.57	39.0
rel-hm	item-sales	Train	0.0	0.0	0.076	87.16
		Val	0.0	0.0	0.086	40.36
		Test	0.0	0.0	0.076	38.31
		Total	0.0	0.0	0.076	87.16
rel-stack	post-votes	Train	0.0	0.0	0.093	78.0
		Val	0.0	0.0	0.062	36.0
		Test	0.0	0.0	0.068	26.0
		Total	0.0	0.0	0.090	78.0
rel-trial	study-adverse	Train	0.0	2.0	39.84	28,085.0
		Val	0.0	2.0	57.08	17,245.0
		Test	0.0	3.0	57.93	5,978.0
		Total	0.0	2.0	42.20	28,085.0
	site-success	Train	0.0	0.0	0.44	1.0
		Val	0.0	0.4	0.47	1.0
		Test	0.0	0.17	0.4	1.06
		Total	0.0	0.0	0.45	1.0
rel-event	user-attendance	Train	0.0	0.0	0.37	16.0
		Val	0.0	0.0	0.28	5.0
		Test	0.0	0.0	0.26	8.0
		Total	0.0	0.0	0.34	16.0
rel-avito	ad-ctr	Train	0.00052	0.018	0.045	1.0
		Val	0.00091	0.018	0.048	1.0
		Test	0.00085	0.019	0.052	1.0
		Total	0.00052	0.018	0.047	1.0

Table 13: RELBENCH link prediction training table link statistics.

Dataset	Task	Split	#Links	Avg #links per entity/timestamp	%Repeated links
rel-amazon	user-item-purchase	Train	11,759,844	2.18	-
		Val	802,540	2.28	0.18
		Test	918,919	2.33	0.15
	user-item-rate	Train	7,146,115	1.8	-
		Val	519,496	2.01	0.19
		Test	599,867	2.05	0.15
	user-item-review	Train	5,138,184	2.19	-
		Val	268,651	2.3	0.18
		Test	305,476	2.4	0.15
rel-hm	user-item-purchase	Train	13,191,321	3.38	-
		Val	237,152	3.18	3.51
		Test	207,996	3.10	3.76
rel-stack	user-post-comment	Train	43,337	2.08	-
		Val	1,603	1.94	3.43
		Test	1,517	2.0	4.09
	post-post-related	Train	7,162	1.2	-
		Val	294	1.3	0.0
		Test	359	1.39	1.39
rel-trial	condition-sponsor-run	Train	503,176	12.51	-
		Val	30,448	14.63	34.48
		Test	25,694	12.49	38.37
	site-sponsor-run	Train	1,485,360	2.27	-
		Val	80,103	2.16	20.91
		Test	50,635	1.85	23.29
rel-avito	user-ad-visit	Train	2,738,733	31.53	-
		Val	877,441	29.27	6.79
		Test	712,985	19.73	4.73

F Dataset Schema

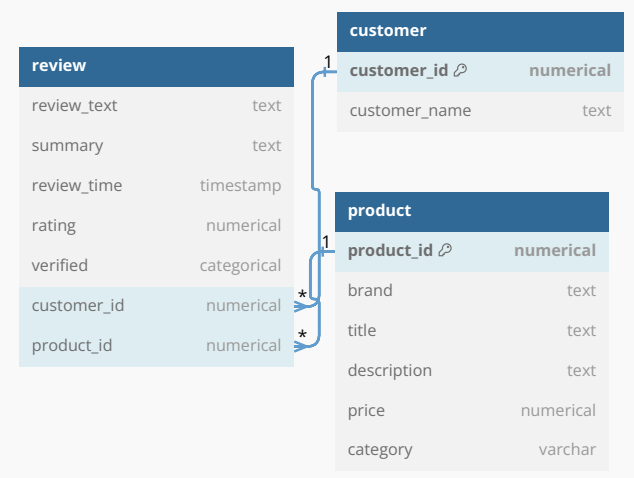


Figure 11: rel-amazon database diagram.

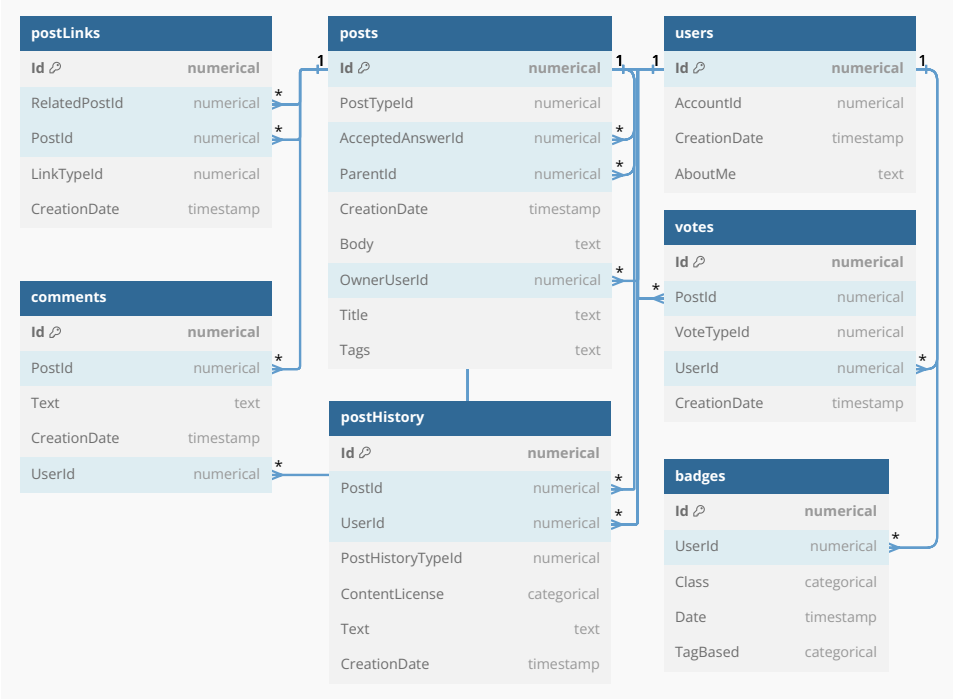


Figure 12: `rel-stack` database diagram.

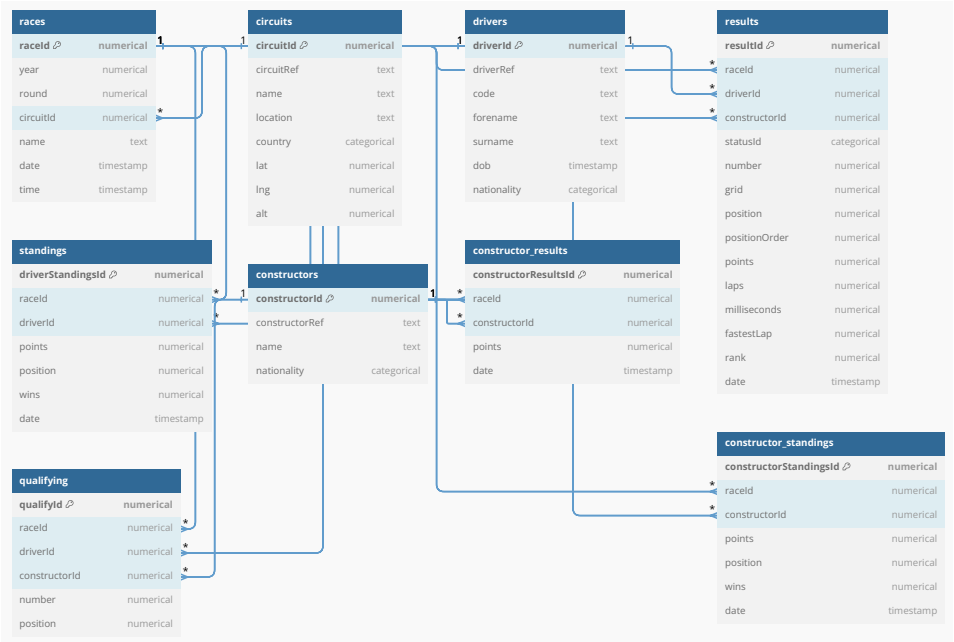


Figure 13: `rel-f1` database diagram.

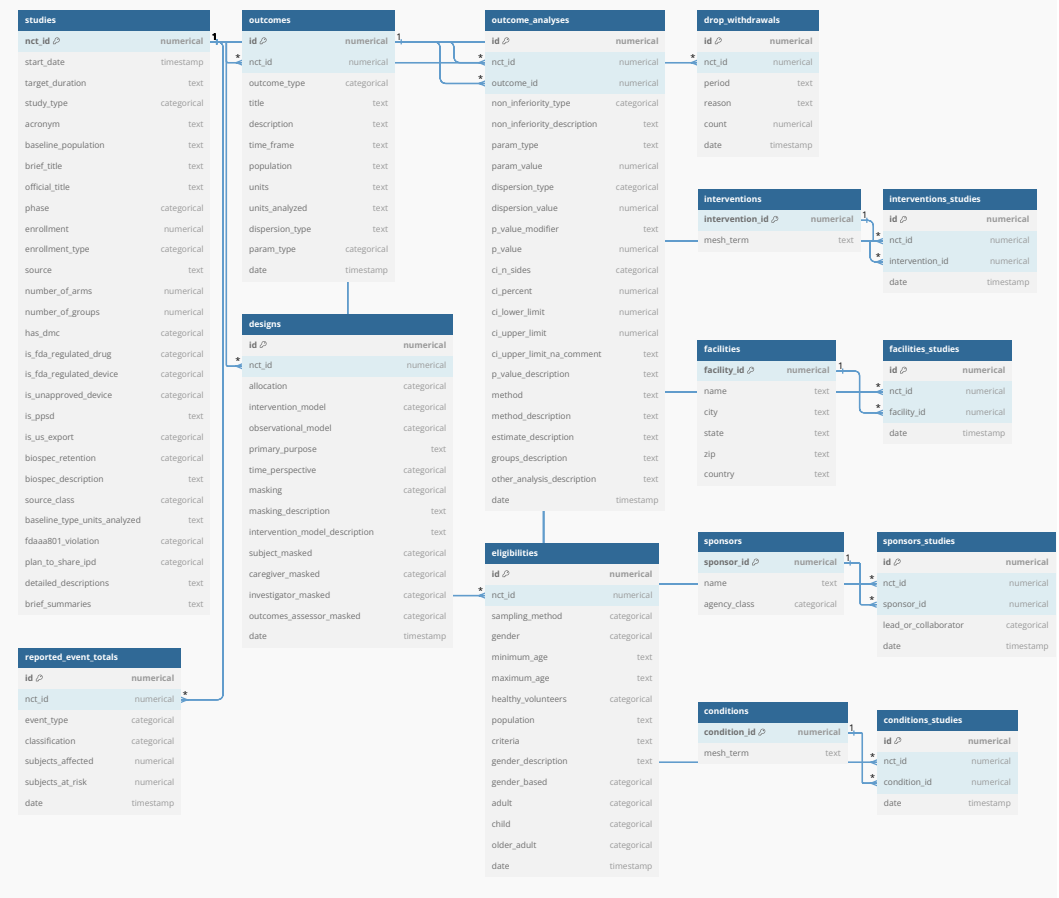


Figure 14: `rel-trial` database diagram.

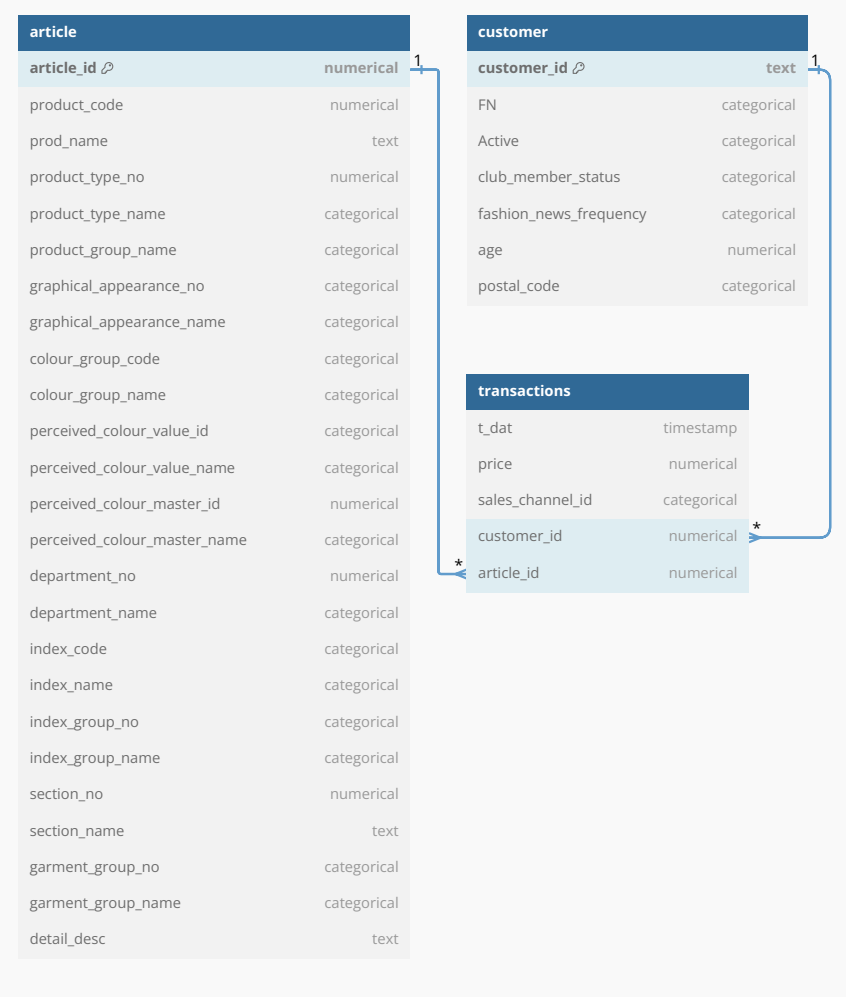


Figure 15: `rel-hm` database diagram.

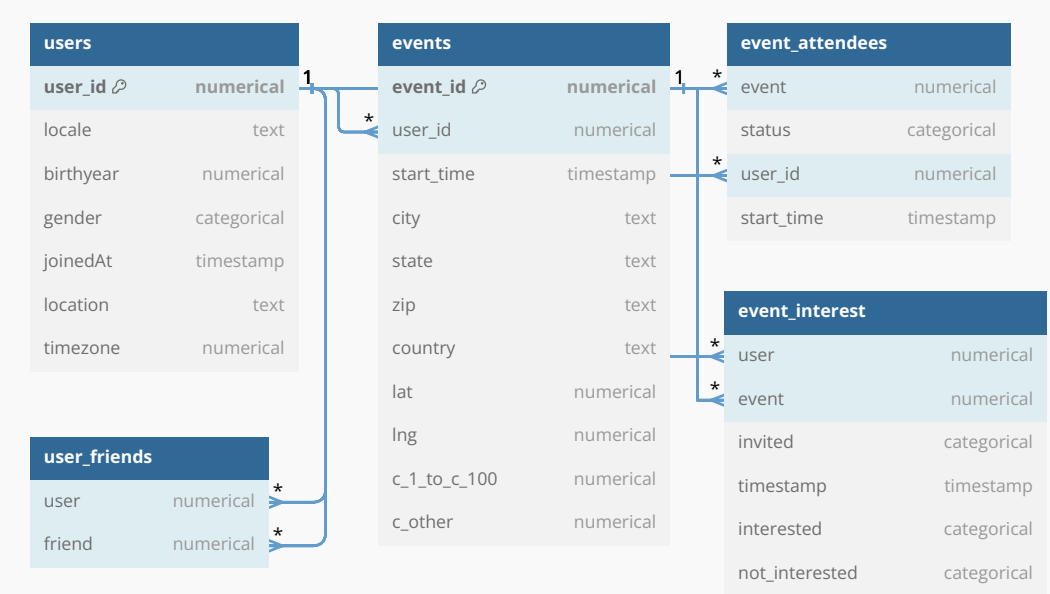


Figure 16: `rel-event` database diagram.

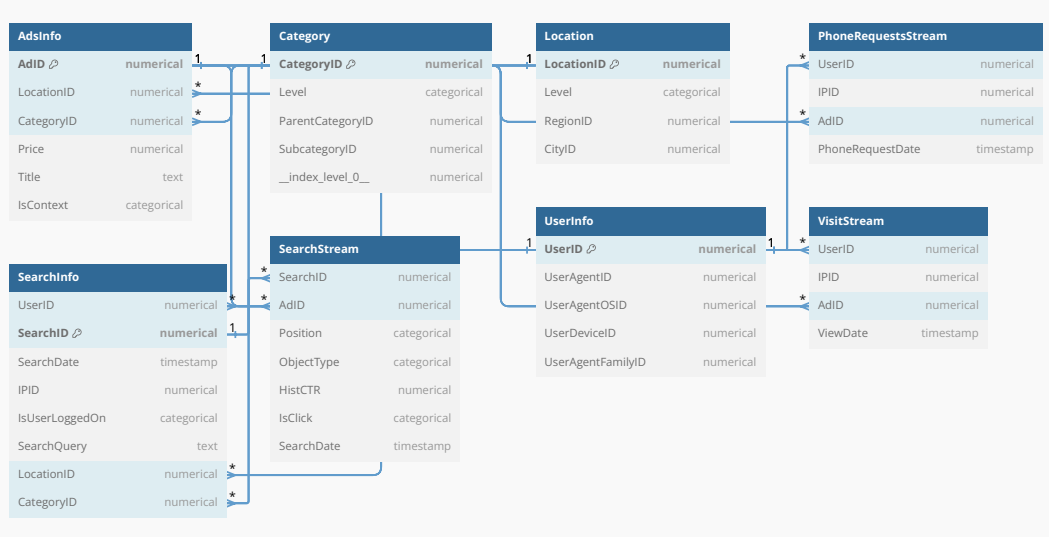


Figure 17: `rel-avito` database diagram.

G Broader Impact

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.

Whilst we strongly believe the RELBENCH has all the ingredients needed to be a long term benchamrk for relational deep learning, there are also possibilities for improvement and extension. Two such possibilities include: (1) RDL at scale: currently our implementation must load the entire database

into working memory during training. For very large datasets this is not viable. Instead, a custom batch sampler is needed that acesses the database via queries to sample specific entities and their pkey-fkey neighbors; (2) Fully inductive link-prediction: our current link-prediction implementation supports predicting links for test time pairs (*head,tail*) where *head* is potentially new (unseen during training) and *tail* seen in the training data. Extending this formulation to be fully inductive (*i.e.*, *tail* unseen during training) is possible, but out of the scope of this work for now.