RELBENCH: A Benchmark for Deep 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

We present RELBENCH, a public benchmark for solving predictive tasks over relational databases with graph neural networks. RELBENCH provides databases and tasks spanning diverse domains and scales, and is intended to be a foundational infrastructure for future research. We use RELBENCH to conduct the first comprehensive study of Relational Deep Learning (RDL) (Fey *et al.*, 2024), which combines graph neural network predictive models with (deep) tabular models that extract initial entity-level representations from raw tables. End-to-end learned RDL models fully exploit the predictive signal encoded in primary-foreign key links, marking a significant shift away from the dominant paradigm of manual feature engineering combined with tabular models. To thoroughly evaluate RDL against this prior gold-standard, we conduct an in-depth user study where an experienced data scientist manually engineers features for each task. In this study, RDL learns better models whilst reducing human work needed by more than an order of magnitude. This demonstrates the power of deep learning for solving predictive tasks over relational databases, opening up many new research opportunities enabled by RELBENCH.

1 Introduction

Relational databases are the most widely used database management system, underpinning much of the digital economy. Their popularity stems from their table storage structure, making maintenance relatively easy, and data simple to access using powerful query languages such as SQL. Because of their popularity, AI systems across a wide variety of domains are built using data stored in relational databases, including e-commerce, social media, banking systems, healthcare, manufacturing, and open-source scientific repositories (Johnson *et al.*, 2016; PubMed, 1996).

Despite the importance of relational databases, the rich relational information is typically foregone, as no model architecture is capable of handling varied database structures. Instead, data is "flattened" into a simpler format such as a single table, often by manual feature engineering, on which standard tabular models can be used (Kaggle, 2022). This results in a significant loss in predictive signal, and creates a need for data extraction pipelines that frequently cause bugs and add to software complexity.

To fully exploit the predictive signal encoded in the relations between entities, a new proposal is to re-cast relational data as an exact *graph* representation, with a node for each entity in the database, edges indicating primary-foreign key links, and node features extracted using deep tabular models, an approach termed *Relational Deep Learning* (RDL) (Fey *et al.*, 2024). The graph representation allows Graph Neural Networks (GNNs) (Gilmer *et al.*, 2017; Hamilton *et al.*, 2017) to be used as predictive models. RDL is the first approach for an end-to-end learnable neural network model with access to all possible predictive signal in a relational databases, and has the potential to unlock new

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

38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks.

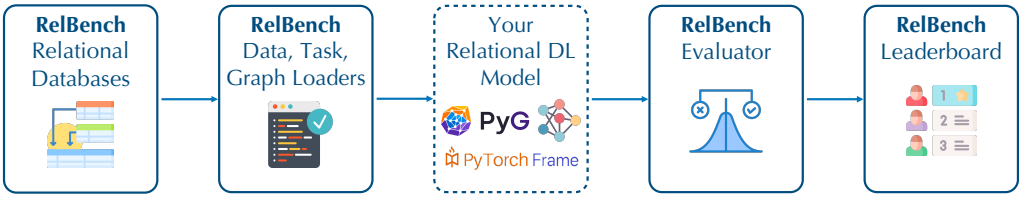

Figure 1: **RELBENCH** enables training and evaluation of deep learning models on relational databases. RELBENCH supports framework agnostic data loading, task specification, standardized data splitting, standardized evaluation metrics, and a leaderboard for tracking progress. RELBENCH also includes a pilot implementation of the relational deep learning blueprint of Fey *et al.* (2024).

levels of predictive power. However, the development of relational deep learning is limited by a complete lack of infrastructure to support research, including: (i) standardized benchmark databases and tasks to compare methods, (ii) initial implementation of RDL, including converting data to graph form and GNN training, and (iii) a pilot study of the effectiveness of relational deep learning.

Here we present RELBENCH, the first benchmark for relational deep learning. RELBENCH is intended to be the foundational infrastructure for future research into relational deep learning, providing a comprehensive set of databases across a variety of domains, including *e-commerce*, *Q&A platforms*, *medical*, and *sports* databases. RELBENCH databases span orders of magnitude in size, from 74K entities to 41M entities, and have very different time spans, between 2 weeks and 55 years of training data. They also vary significantly in their relational structure, with the total number of tables varying between 3 and 15, and total number of columns varying from 15 to 140. Each database comes with multiple predictive tasks, 30 in total, including entity classification/regression and recommendation tasks, each chosen for their real-world significance.

In addition to databases and tasks, we release open-source software designed to make relational deep learning widely available. This includes (i) the RELBENCH Python package for easy database and task loading, (ii) the first open-source implementation of relational deep learning, designed to be easily modified by researchers, and (iii) a public leaderboard for tracking progress. We comprehensively benchmark our initial RDL implementation on all RELBENCH tasks, comparing to various baselines.

The most important baseline we compare to is a strong "data scientist" approach, for which we recruited an experienced individual to solve each task by manually engineering features and feeding them into tabular models. This approach is the current gold-standard for building predictive models on relational databases. The study, which we open source for reproducibility, finds that RDL models match or outperform the data scientist's models in accuracy, whilst reducing human hours worked by 96%, and lines of code by 94% on average. This constitutes the first empirical demonstration of the central promise of RDL, and points to a long-awaited end-to-end deep learning solution for relational data.

Our website[2] is a comprehensive entry point to RDL, describing RELBENCH databases and tasks, access to code on GitHub, the full relational deep learning blueprint, and tutorials for adding new databases and tasks to RELBENCH to allow researchers to experiment with their problems of interest.

2 Overview and Design

RELBENCH provides a collection of diverse real-world **relational databases** along with a set of realistic **predictive tasks** associated with each database. Concretely, we provide:

- **Relational databases**, consisting of a set of tables connected via primary-foreign key relationships. Each table has columns storing diverse information about each entity. Some tables also come with *time columns*, indicating the time at which the entity is created (*e.g.*, transaction date).
- **Predictive tasks over a relational database**, which are defined by a **training table** (Fey *et al.*, 2024) with columns for Entity ID, seed time, and target labels.The seed time indicates *at which time* the target is to be predicted, filtering future data.

Next we outline key design principles of RELBENCH with an emphasis on data curation, data splits, research flexibility, and open-source implementation.

[2] https://relbench.stanford.edu.

Table 1: **Statistics of RELBENCH datasets.** Datasets vary significantly in the number of tables, total number of rows, and number of columns. In this table, we only count rows available for test inference, i.e., rows upto the test time cutoff.

Name	Domain	#Tasks	Tables			Timestamp (year-mon-day)		
			#Tables	#Rows	#Cols	Start	Val	Test
rel-amazon	E-commerce	7	3	15,000,713	15	2008-01-01	2015-10-01	2016-01-01
rel-avito	E-commerce	4	8	20,679,117	42	2015-04-25	2015-05-08	2015-05-14
rel-event	Social	3	5	41,328,337	128	1912-01-01	2012-11-21	2012-11-29
rel-f1	Sports	3	9	74,063	67	1950-05-13	2005-01-01	2010-01-01
rel-hm	E-commerce	3	3	16,664,809	37	2019-09-07	2020-09-07	2020-09-14
rel-stack	Social	5	7	4,247,264	52	2009-02-02	2020-10-01	2021-01-01
rel-trial	Medical	5	15	5,434,924	140	2000-01-01	2020-01-01	2021-01-01
Total		30	51	103,466,370	489	/	/	/

Data Curation. Relational databases are widespread, so there are many candidate predictive tasks. For the purpose of benchmarking we carefully curate a collection of relational databases and tasks chosen for their rich relational structure and column features. We also adopt the following principles:

- **Diverse domains:** To ensure algorithms developed on RELBENCH will be useful across a wide range of application domains, we select real-world relational databases from diverse domains.
- **Diverse task types:** Tasks cover a wide range of real-world use-cases, including three representative task types: entity classification, entity regression, and recommendation.

RELBENCH databases are summarized in Table 1, covering E-commerce, social, medical, and sports domains. The databases vary significantly in the numbers of rows (*i.e.*, data scale) the number of columns and tables, as well as the time ranges of the databases. Tasks are summarized in Table 2, each corresponding to a predictive problem of practical interest such as predicting customer churn, predicting the number of adverse events in a clinical trial, and recommending posts to users.

Data Splits. Data is split temporally, with models trained on rows up to VAL_TIMESTAMP, validated on the rows between VAL_TIMESTAMP and TEST_TIMESTAMP, and tested on the rows after TEST_TIMESTAMP. Our implementation carefully hides data after TEST_TIMESTAMP during inference to systematically avoid test time data leakage (Kapoor and Narayanan, 2023), and uses an elegant solution proposed by Fey *et al.* (2024) to avoid time leakage during training and validation through temporal neighbor sampling. In general, it is the designers responsibility to avoid time leakage. We recommend using our carefully tested implementation where possible.

Research Flexibility. RELBENCH is designed to allow significant freedom in future research directions. For example, RELBENCH tasks share the same (VAL_TIMESTAMP and TEST_TIMESTAMP) splits across tasks within the same relational database. This opens up exciting opportunities for multi-task learning and pre-training to simultaneously improve different predictive tasks within the same relational database. We also expose the logic for converting databases into graphs. This allows future work to consider modified graph constructions, or creative uses of the raw data.

Open-source RDL Implementation. As well as datasets and tasks, we provide the first open-source implementation of relational deep learning. See Figure 2 of Fey *et al.* (2024) for a high-level overview. A neural network is learned over a heterogeneous temporal graph that exactly represents the database in order to make prediction over nodes (for entity classification and regression) and links (for recommendation). Our implementation is built on top of PyTorch Frame (Hu *et al.*, 2024) for extracting initial node embeddings from raw table features, and PyTorch Geometric (Fey and Lenssen, 2019) for GNN modeling. See Section A for details.

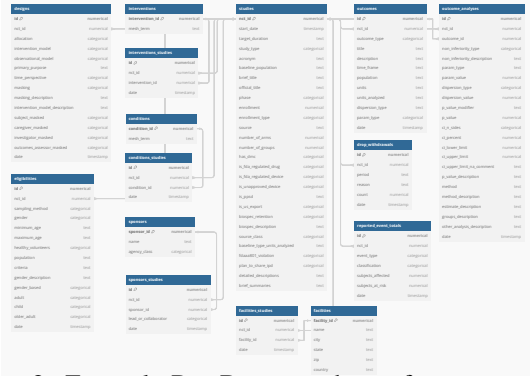

Figure 2: Example RELBENCH schema for `rel-trial` database. RELBENCH databases have complex relational structure and rich column features.

Table 2: Full list of predictive tasks for each RELBENCH dataset (introduced in Table 1).

Dataset	Task name	Task type	#Rows of training table			#Unique	%train/test	#Dst
			Train	Validation	Test	Entities	Entity Overlap	Entities
rel-amazon	user-churn	entity-cls	4,732,555	409,792	351,885	1,585,983	88.0	—
	item-churn	entity-cls	2,559,264	177,689	166,842	416,352	93.1	—
	user-ltv	entity-reg	4,732,555	409,792	351,885	1,585,983	88.0	—
	item-ltv	entity-reg	2,707,679	166,978	178,334	427,537	93.5	—
	user-item-purchase	recommendation	5,112,803	351,876	393,985	1,632,909	87.4	12,562,384
	user-item-rate	recommendation	3,667,157	257,939	292,609	1,481,360	81.0	7,665,611
	user-item-review	recommendation	2,324,177	116,970	127,021	894,136	74.1	5,406,835
rel-avito	ad-ctr	entity-reg	5,100	1,766	1,816	4,997	59.8	—
	user-clicks	entity-cls	59,454	21,183	47,996	66,449	45.3	—
	user-visits	entity-cls	86,619	29,979	36,129	63,405	64.6	—
	user-ad-visit	recommendation	86,616	29,979	36,129	63,402	64.6	3,616,174
rel-event	user-attendance	entity-reg	19,261	2,014	2,006	9,694	14.6	—
	user-repeat	entity-cls	3,842	268	246	1,514	11.5	—
	user-ignore	entity-cls	19,239	4,185	4,010	9,799	21.1	—
rel-f1	driver-dnf	entity-cls	11,411	566	702	821	50.0	—
	driver-top3	entity-cls	1,353	588	726	134	50.0	—
	driver-position	entity-reg	7,453	499	760	826	44.6	—
rel-hm	user-churn	entity-cls	3,871,410	76,556	74,575	1,002,984	89.7	—
	item-sales	entity-reg	5,488,184	105,542	105,542	105,542	100.0	—
	user-item-purchase	recommendation	3,878,451	74,575	67,144	1,004,046	89.2	13,428,473
rel-stack	user-engagement	entity-cls	1,360,850	85,838	88,137	88,137	97.4	—
	user-badge	entity-cls	3,386,276	247,398	255,360	255,360	96.9	—
	post-votes	entity-reg	2,453,921	156,216	160,903	160,903	97.1	—
	user-post-comment	recommendation	21,239	825	758	11,453	59.9	44,940
	post-post-related	recommendation	5,855	226	258	5,924	8.5	7,456
rel-trial	study-outcome	entity-cls	11,994	960	825	13,779	0.0	—
	study-adverse	entity-reg	43,335	3,596	3,098	50,029	0.0	—
	site-success	entity-reg	151,407	19,740	22,617	129,542	42.0	—
	condition-sponsor-run	recommendation	36,934	2,081	2,057	3,956	98.4	533,624
	site-sponsor-run	recommendation	669,310	37,003	27,428	445,513	48.3	1,565,463

3 RELBENCH Datasets

RELBENCH contains 7 datasets each
with rich relational structure, providing a challenging environment for developing and comparing relational deep learning methods (see Figure 2 for an example). The datasets are carefully processed from real-world relational databases and span diverse domains and sizes. Each database is associated with multiple individual predictive tasks defined in Section 4. Detailed statistics of each dataset can be found in Table 1. We briefly describe each dataset.

rel-amazon. The Amazon E-commerce database records products, users, and reviews across Amazon's E-commerce platform. It contains rich information about products and reviews. Products include the price and category of each, reviews have the 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.

rel-f1. The F1 database tracks all-time Formula 1 racing data and statistics since 1950. It provides detailed information for various stakeholders including drivers, constructors, engine manufacturers, and tyre manufacturers. Highlights include data on all circuits (*e.g.*geographical details), and full historical data from every season. This includes overall standings, race results, and more specific data like practice sessions, qualifying positions, sprints, and pit stops.

rel-stack. Stack Exchange is a network of question-and-answer websites on different topics, where questions, answers, and users are subject to a reputation award process. The reputation system allows the sites to be self-moderating. The database includes detailed records of activity including user biographies, posts and comments (with raw text), edit histories, voting, and related posts. In our benchmark, we use the stats-exchange site.

rel-trial. The clinical trial database is curated from AACT initiative, which consolidates all protocol and results data from studies registered on ClinicalTrials.gov. It offers extensive information about clinical trials, including study designs, participant demographics, intervention details, and outcomes. It is an important resource for health research, policy making, and therapeutic development.

rel-hm. The H&M relational database hosts extensive customer and product data for online shopping experiences across its extensive network of brands and stores. This database includes

detailed customer purchase histories and a rich set of metadata, encompassing everything from basic demographic information to extensive details about each product available.

rel-event. The Event Recommendation database is obtained from user data on a mobile app called Hangtime. This app allows users to keep track of their friends' social plans. The database contains data on user actions, event metadata, and demographic information, as well as users' social relations, which captures how social relations can affect user behavior. Data is fully anonymized, with no personally identifiable information (such as names or aliases) available.

rel-avito. Avito is a leading online advertisement platform, providing a marketplace for users to buy and sell a wide variety of products and services, including real estate, vehicles, jobs, and goods. The Avito Context Ad Clicks dataset on Kaggle is part of a competition aimed at predicting whether an ad will be clicked based on contextual information. This dataset includes user searches, ad attributes, and other related data to help build predictive models.

Data Provenance. All data is sourced from publicly available repositories with licenses permitting usage for research purposes. See Appendix E for details of data sources, licenses, and more.

4 Predictive Tasks on RELBENCH Datasets

RELBENCH introduces 30 new predictive tasks defined over the databases introduced in Section 2. A full list of tasks is given in Table 2, with high-level descriptions given in Appendix B (and our website) due to space limitations. 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.

Table 3: Entity classification results (AUROC, higher is better) on RELBENCH. Best values are in bold. See Table 6 in Appendix C 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 %

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.

Experimental results. Results are given in Table 3, 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.

Table 4: Entity regression results (MAE, lower is better) on RELBENCH. Best values are in bold. See Table 7 in Appendix C 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 %

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 4 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 D, 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 B 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 5: Recommendation results (MAP, higher is better) on RELBENCH. Best values are in bold. See Table 8 in Appendix C 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 5. 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 test RDL in the most challenging circumstances possible, we undertake a human trial wherein a data scientist solves each task by manually designing features and feeds them into tabular methods such at LightGBM or XGBoost (Chen and Guestrin, 2016; Ke *et al.*, 2017). This represents the prior gold-standard for building predictive models on relational databases (Heaton, 2016), and the key point of comparison for RDL.

We structure our user study along the five main data science workflow steps:

1. **Exploratory data analysis (EDA):** Explore the dataset and task to understand its characteristics, including what column features there are, and if there is any missing data.
2. **Feature ideation:** Based on EDA and intuition from prior experiences, propose a set of entity-level features that the data scientist believes may contain predictive signal for the task.
3. **Feature enginnering:** Using query languages such as SQL to compute the proposed features, and add them as extra columns to the target table of interest.
4. **Tabular ML:** Run tabular methods such as LightGBM or XGBoost on the table with extra features to produce a predictive model, and record the test performance.
5. **Post-hoc analysis of feature importance (Optional):** Common tools include SHAP and LIME, which aim to explain the contribution of each input feature to the final performance.

Consider for example the `rel-hm` dataset (schema in Appendix E) and the task of predicting customer churn. Here the CUSTOMER table only contains simple biographical information such as username and joining date. To capture more predictive information, additional features, such as *time since last purchase*, can be computed using the other tables, and added to the CUSTOMER table. We give a detailed walk-through of the data scientist's work process for solving this specific task in Appendix D. We strongly encourage the interested reader to review this, as it highlights the significant amount of task-specific effort that this workflow necessitates.

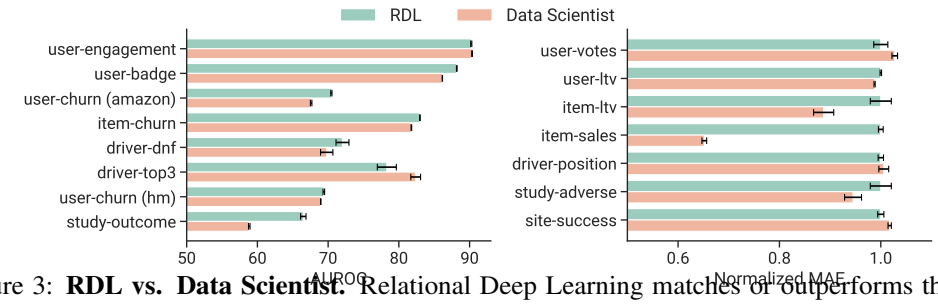

Figure 3: **RDL vs. Data Scientist.** Relational Deep Learning matches or outperforms the data scientist in 11 of 15 tasks. Left shows entity classification AUROC, right shows entity regression, reporting MAE normalized so that the RDL MAE is always 1.

Limitations of Manual Feature Engineering. This workflow suffers from several fundamental limitations. Most obviously, since features are hand designed they only capture part of the predictive signal in the database, useful signal is easily missed. Additionally, feature complexity is limited by human reasoning abilities, meaning that higher-order interactions between entities are often overlooked. Beyond predictive signal, the other crucial limitation of feature engineering is its extremely manual nature—every time a new model is built a data scientist has to repeat this process, requiring many hours of human labor, and significant quantities of new SQL code to design features (Zheng and Casari, 2018). Our RDL models avoid these limitations (see Section 5).

Data Scientist. To conduct a thorough comparison to this process, we recruit a high-end data scientist with Stanford CS MSc degree, 4.0 GPA, and 5 years of experience of building machine learning models in the financial industry. This experience includes a significant amount of time building machine learning models in exactly above five steps, as well as broader data science expertise.

User Study Protocol.

Because of the open-ended nature of feature engineering and model development, we follow a specific protocol for the user study in order to standardize the amount of effort dedicated to each dataset and task. Tracking the 5 steps outlined above, we impose the following rules:

1. **EDA:** The time allotted for data exploration is capped at 4 hours. This threshold was chosen to give the data scientist enough time to familiarize themselves with the schema, visualize key relationships and distributions, and take stock of any outliers in the dataset, while providing a reasonable limit to the effort applied.

2. **Feature ideation:** Feature ideation is performed manually with pen and paper, and is limited to 1 hour. In practice, the data scientist found that 1 hour was plenty of time to enumerate all promising features at that time, especially since many ideas naturally arise during the EDA process already.

3. **Feature engineering:** The features described during the ideation phase are then computed using SQL queries. The time taken to write SQL code to generate the features is unconstrained in order to eliminate code writing speed as a factor in the study. We do, however, record code writing time for our timing benchmarking. This stage presented the most variability in terms of time commitment, partly because it is unconstrained, but mostly because the implementation complexity of the features itself is highly variable.

4. **Tabular ML:** For tabular ML training, we provide a standardized LightGBM training script including comprehensive hyperparameter tuning. The data scientist needs only to feed the table full of engineered features into this training script, which returns test performance results. However, there is some non-trivial amount of work required to transform the output of the SQL queries from the previous section into the Python objects (arrays) required for training LightGBM. Again, the time taken for this additional pre-preocessing is recorded.

5. **Post-hoc analysis of feature importance:** Finally, after successfully training a model, an evaluation of model predictions and feature importance is carried out. This mostly serves as a general sanity check and an interesting corollary of the data scientist's work that provides task-specific insights (see Appendix D). In practice, this took no more than a few minutes per task and this time was not counted toward the total time commitment.

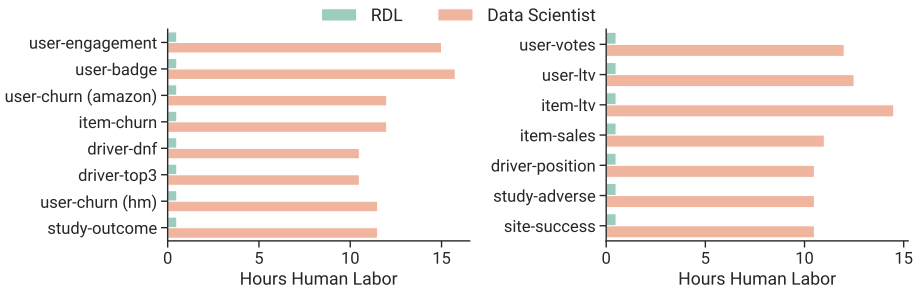

Figure 4: **RDL vs. Data Scientist.** Relational Deep Learning reduces the hours of human work required to solve a new task by 96% on average (from 12.3 to 0.5 hours). Left shows node-level classification, right shows node-level regression.

Reproducibility. All of the data scientist's workings are released[3] to ensure reproducibility and demonstrate the significant lengths gone through to build as accurate models as possible. In Appendix D we walk through a complete example for a single dataset and task, showing the data-centric insights it yields. An important by-product is a close analysis of which features contribute to model performance, which we believe will help inspire future well-motivated RDL research directions.

Results. As well as (i) raw predictive power, we compare the data scientist to our RDL models in terms of (ii) hours of human work, and (iii) number of new lines of code required to solve each task. We measure the *marginal* effort, meaning that we do not include code infrastructure that is reused across tasks, including for example data loading logic and training scripts for RDL or LightGBM models. Accordingly, we only compare model development, not data preparation/loading. Indeed the data loading pipeline is shared between RDL and the data scientist, so RDL does not introduce any significant overheads for data loading/preparation over a data scientist's approach. We believe that accelerating model development (apart from data loading) is valuable in many use cases where engineers need to solve many different predictive tasks over a single database.

Summary. Figures 3, 4, and 5 show that RDL learns highly predictive models, outperforming the data scientist in 11 of 15 tasks, whilst reducing hours worked by 96% on average, and lines of code by 94% on average. On average, it took the data scientist 12.3 hours to solve each task using traditional feature engineering. By contrast it takes roughly 30 minutes to solve a task with RDL. This observation is *the* central value proposition of relational deep learning, pointing the way to unlocking new levels of predictive power, and potentially a new economic model for solving predictive tasks on relational databases. Replacing hand-crafted solutions with end-to-end learnable models has been a key takeaway from the last 15 years of AI research. It is therefore remarkable how little impact deep learning has had on ML on relational databases, one of the most widespread applied ML use cases. To the best of our knowledge, is RDL the first deep learning approach for relational databases that has demonstrated efficacy compared with established data science workflows. We highlight that all RELBENCH tasks were solved with a single set of default hyperparameters (with 2 exceptions requiring small modifications to learning rate, number of epochs, and GNN aggregation function). This demonstrates the robustness of RDL, and that the performance of RDL in Figure 3 is not due to extensive hyperparamter search. Indeed, the single set of RDL hyperparameters is compared to a carefully tuned LightGBM, which was allowed to search over 10 sets of hyperparameters.

Predictive Power. Results shown in Figures 3. Whilst outperforming the data scientist in 11 of 15 tasks, we note that RDL best outperforms the data scientist on classification tasks, struggling more on regression. Indeed it was necessary for us to apply a "boosting" to the RDL model to improve performance (see Appendix D). Even with boosting, the data scientist model outperforms RDL in several cases. One cause we identify is that the MLP output head of the GNN is poorly suited to regression tasks (see Appendix D). This suggests an opportunity for improved output heads for regression tasks. We stress that our RDL implementation is an *initial* demonstration. We believe there is significant scope for new research leading to large improvements in performance. In particular, ideas from graph ML, deep tabular ML, and time-series modeling are well suited to advance RDL.

Human Work. Results shown in Figure 4. In our user study RDL required 96% less hours work to solve a new task, compared to the data scientist work flow. The RDL solutions always took less

[3]See https://github.com/snap-stanford/relbench-user-study.

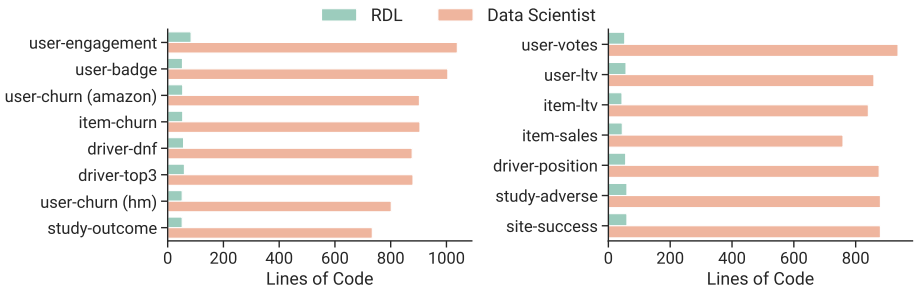

Figure 5: **RDL vs. Data Scientist.** Relational Deep Learning reduces the new lines of code needed to solve a new task by 94%. Left shows entity classification, right shows entity regression.

than an hour to write, whilst the data scientist took 12 hours on average, with a standard deviation of 1.6 hours. We emphasize that this measures *marginal* effort, i.e., it does not include reusable code that can be amortized over many tasks. RDL compares favorably to data scientist because a large majority of RDL code is reusable for new tasks (a GNN architecture and training loop needs only to be defined once) whereas a large portion of the data scientist's code is task specific and must be re-done afresh for every new task that needs to be solved.

Lines of Code. Results shown in Figure 5. For the RDL model, the only new addition needed to solve a new task is the code describing how to compute the training supervision for the RDL, which is stored in the training table. This requires a similar number of lines of code for each task, with 56 lines of code on average, with standard deviation 8.8, with the data scientist requiring with 878 ± 77. The minimum lines of code required by RDL is 44, compared to 734 for the data scientist, and maximum is 84 compared to 1039 for the data scientist. Examples of the RDL code for `rel-amazon` tasks can be viewed here. We record the number of lines of data scientist code for EDA and SQL files, and the manipulations needed to format data to be fed into the pre-prepared LightGBM script.

6 Related Work

Graph Machine Learning Benchmarks. Challenging and realistic benchmarks drive innovation in methodology. A classic example is the ImageNet (Deng *et al.*, 2009), introduced prior to the rise of deep learning, which was a key catalyst for the seminal work of Krizhevsky *et al.* (2017). In graph machine learning, benchmarks such as the Open Graph Benchmark (Hu *et al.*, 2020), TUDataset (Morris *et al.*, 2020), and more recently, the Temporal Graph Benchmark (Huang *et al.*, 2024) have sustained the growth and maturation of graph machine learning as a field. RELBENCH differs since instead of collecting together tasks are already recognized as graph machine learning tasks, RELBENCH presents existing tasks typically solved using other methods, as graph ML tasks. As a consequence, RELBENCH significantly expands the space of problems solvable using graph ML. Whilst graph ML is a key part of this benchmark, relational deep learning is a *new problem*, requiring only need good GNNs, but also innovation on tabular learning to fuse multimodal input data with the GNN, temporal learning, and even graph construction. We believe that advancing the state-of-the-art on RELBENCH will involve progress in all of these directions.

Relational Deep Learning. Several works have proposed to use graph neural networks for learning on relational data (Schlichtkrull *et al.*, 2018; Cvitkovic, 2019; Šír, 2021; Zahradník *et al.*, 2023). They explored different graph neural network architectures on (heterogeneous) graphs, leveraging relational structure. Recently, Fey *et al.* (2024) proposed a general end-to-end learnable framework for solving predictive tasks on relational databases, treating temporality as a core concept.

7 Conclusion

We introduce RELBENCH, a benchmark for 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.

References

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.

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

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

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.

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

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.

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

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

Weihua Hu, Yiwen Yuan, Zecheng Zhang, Akihiro Nitta, Kaidi Cao, Vid Kocijan, Jure Leskovec, and Matthias Fey. Pytorch frame: A modular framework for multi-modal tabular learning. *arXiv preprint arXiv:2404.00776*, 2024.

Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. *Advances in Neural Information Processing Systems*, 36, 2024.

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.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, 2017.

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

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

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/.

Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. *arXiv preprint arXiv:1205.2618*, 2012.

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.

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

Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *Proceedings of the 42nd international ACM SIGIR conference on Research and development in Information Retrieval*, pages 165–174, 2019.

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.

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

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

A Relational Deep Learning Implementation

As part of RELBENCH, we provide an initial implementation of relational deep learning, based on the blueprint of Fey *et al.* (2024).[4] Our implementation consists four major components: (1) heterogeneous temporal graph, (2) deep learning model, (3) temporal-aware training of the model, and (4) task-specific loss, which we briefly discuss now.

Heterogeneous temporal graph. Given a set of tables with primary-foreigh key relations between them we follow Fey *et al.* (2024) to automatically construct a heterogeneous temporal graph, where each table represents a node type, each row in a table represents a node, and a primary-foreign-key relation between two table rows (nodes) represent an edge between the respective nodes. Some node types are associated with time attributes, representing the timestamp at which a node appears. The heterogeneous temporal graph is represented as a PyTorch Geometric graph object. Each node in the heterogeneous graph comes with a rich feature derived from diverse columns of the corresponding table. We use Tensor Frame provided by PyTorch Frame (Hu *et al.*, 2024) to represent rich node features with diverse column types, *e.g.*, numerical, categorical, timestamp, and text.

Deep learning model. First, we use deep tabular models that encode raw row-level data into initial node embeddings using PyTorch Frame (Hu *et al.*, 2024) (specifically, we use the ResNet tabular model (Gorishniy *et al.*, 2021)). These initial node embeddings are then fed into a GNN to iteratively update the node embeddings based on their neighbors. For the GNN we use the heterogeneous version of the GraphSAGE model (Hamilton *et al.*, 2017; Fey and Lenssen, 2019) with sum-based neighbor aggregation. Output node embeddings are fed into task-specific prediction heads and are learned end-to-end.

Temporal-aware subgraph sampling. We perform temporal neighbor sampling, which samples a subgraph around each entity node at a given seed time. Seed time is the time in history at which the prediction is made. When collecting the information to make a prediction at a given seed time, it is important for the model to only use information from before the seed time and thus not learn from the future (post the seed time). Crucially, when sampling mini-batch subgraphs we make sure that all nodes within the sampled subgraph appear before the seed time (Hamilton *et al.*, 2017; Fey *et al.*, 2024), which systematically avoids time leakage during training. The sampled subgraph is fed as input to the GNN, and trained to predict the target label.

Task-specific prediction head and loss. For entity-level classification, we simply apply an MLP on an entity embedding computed by our GNN to make prediction. For the loss function, we use the binary cross entropy loss for entity classification and L_1 loss for entity regression.

Recommendation requires computing scores between pairs of source nodes and target nodes. For this task type, we consider two representative predictive architectures: two-tower GNN (Wang *et al.*, 2019) and identity-aware GNN (ID-GNN) (You *et al.*, 2021). First, the two-tower GNN computes the pairwise scores via inner product between source and target node embeddings, and the standard Bayesian Personalized Ranking loss (Rendle *et al.*, 2012) is used to train the two-tower model (Wang *et al.*, 2019). Second, the ID-GNN computes the pairwise scores by applying an MLP prediction head on target entity embeddings computed by GNN for each source entity. The ID-GNN is trained by the standard binary cross entropy loss.

B 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.

[4] Code available at: https://github.com/snap-stanford/relbench.

(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.

Table 6: 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	$\mathbf{70.09}_{\pm 1.41}$	$\mathbf{68.60}_{\pm 1.01}$

(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.

C Experiment Details and Additional Results

C.1 Detailed Results

Tables 6, 7 and 8 show mean and standard deviations over 5 runs for the entity classification, entity regression and link prediction results respectively.

C.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 9. 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

Table 7: 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 8: 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	$8.40_{\pm 0.70}$	$10.70_{\pm 1.10}$	$\mathbf{19.00}_{\pm 0.12}$

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.

C.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 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.

Table 9: 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

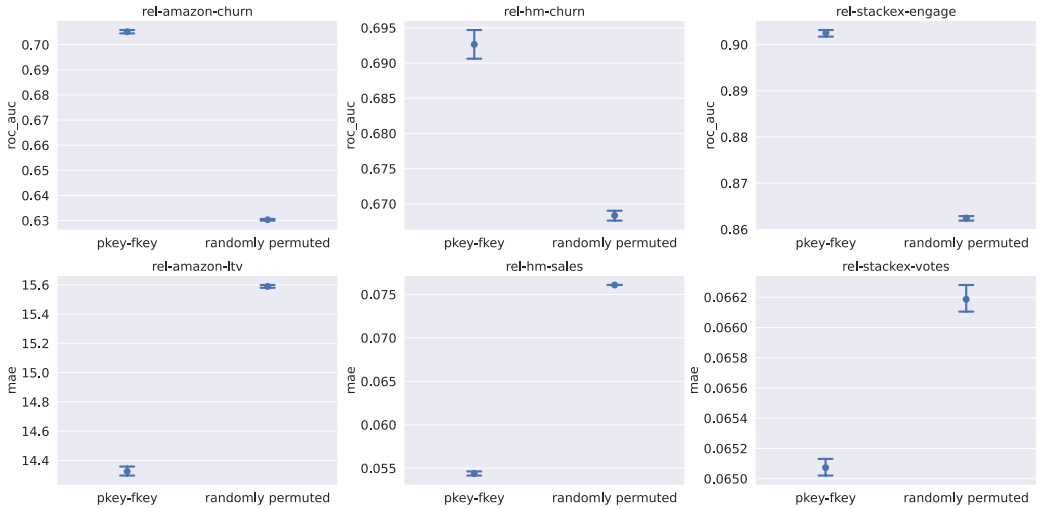

Figure 6: Investigation on the role of leveraging primary-foreign key (pkey-fkey) edges for the GNN. At the top row are three node classification tasks with metric AUROC (higher is better) while at the bottom are three node regression tasks with metric MAE (lower is better), evaluated on the test set. 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.

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. 6 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. 7, 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. 8, 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

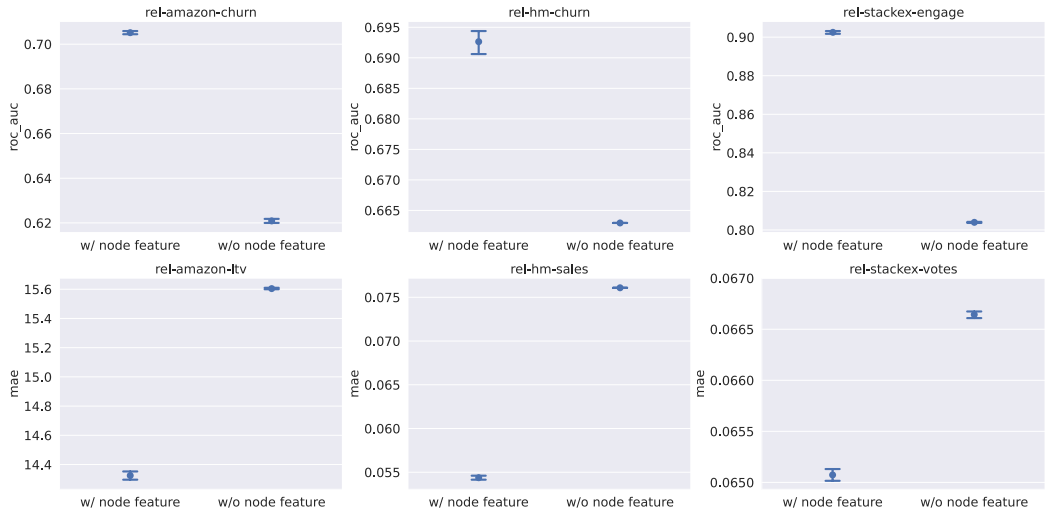

Figure 7: Investigation on the role of node features. At the top row are three node classification tasks with metric AUROC (higher is better) while at the bottom are three node regression tasks with metric MAE (lower is better), evaluated on the test set. We observe that leveraging node features is important for GNN. Error bars correspond to 95% confidence interval.

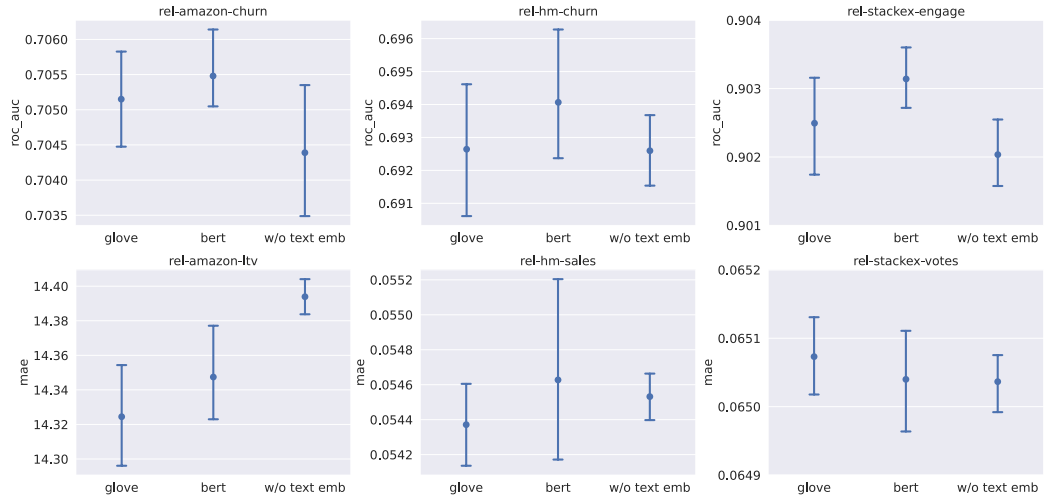

Figure 8: Investigation on the role of text embedding. At the top row are three node classification tasks with metric AUROC (higher is better) while at the bottom are three node regression tasks with metric MAE (lower is better), evaluated on the test set. We observe that adding text embedding using GloVe (Pennington *et al.*, 2014) or BERT (Devlin *et al.*, 2018) generally helps improve the performance. Error bars correspond to 95% confidence interval.

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. 9. 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.

D User Study Additional Details

D.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

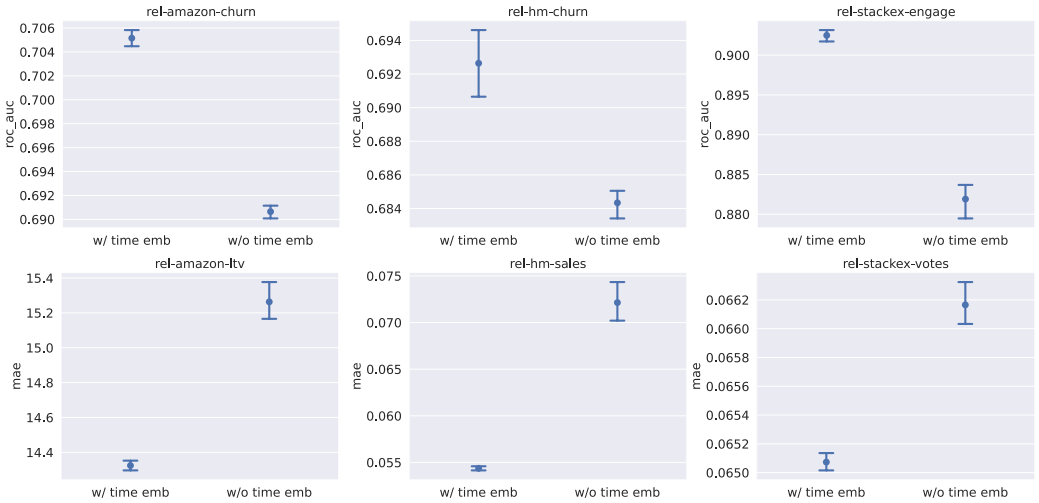

Figure 9: Investigation on the role of time embedding. At the top row are three node classification tasks with metric AUROC (higher is better) while at the bottom are three node regression tasks with metric MAE (lower is better), evaluated on the test set. We find that adding time embedding to the GNN consistently boosts the performance. Error bars correspond to 95% confidence interval.

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).

D.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 10 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.

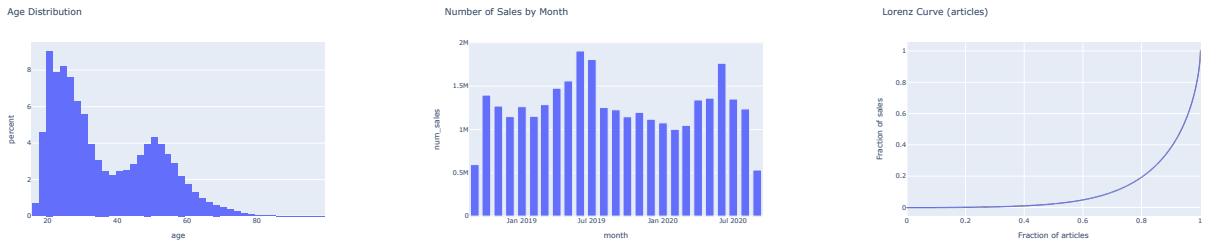

Figure 10: **EDA Plots.** Each plot explores different characteristics of the dataset. Understanding the data and identifying relationships between different quantities is an essential prerequisite to meaningful feature engineering.

  - 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).

D.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.

D.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[5] with some Jinja templating[6] 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).

[5] See https://duckdb.org/.

[6] See https://jinja.palletsprojects.com/en/3.1.x/intro/.

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.

D.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.

D.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[7]. Figure 11 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.

D.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. 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).

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

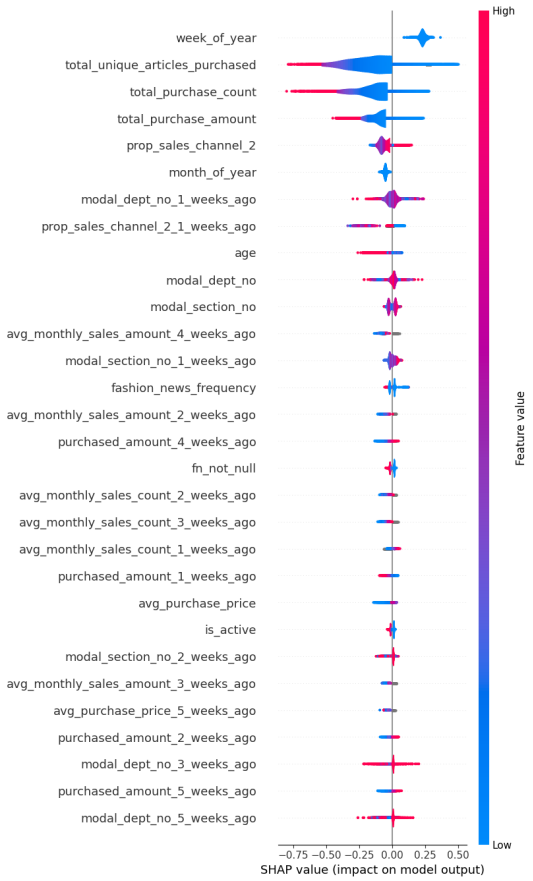

Figure 11: **Feature Importances.** SHAP values of top 30 features ranked by importance. Note: *week_of_year* feature shows little variability because the validation set is temporally concentrated in a few weeks.

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 10, 11, 12 for the results of each of these experiments.

In Figure 3, for regression tasks we report the RDL results using GNN learned features with LightGBM output head. In Table 4 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

Table 10: 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 11: 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	$88.3_{\pm 0.04}$	$\mathbf{88.4}_{\pm 0.04}$

10, 11, 12 provide clear evidence that there is an opportunity for improvements and simplifications, which we leave to future work.

Table 12: 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	$\mathbf{70.61}_{\pm 0.06}$	$70.29_{\pm 0.06}$

E 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 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.

F Additional Training Table Statistics

We report additional training table statistics for all tasks, separated into entity classification (*cf.* Table 13), entity regression (*cf.* Table 14), and link prediction (*cf.* Table 15).

Table 13: 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 14: 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 15: 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

G Dataset Schema

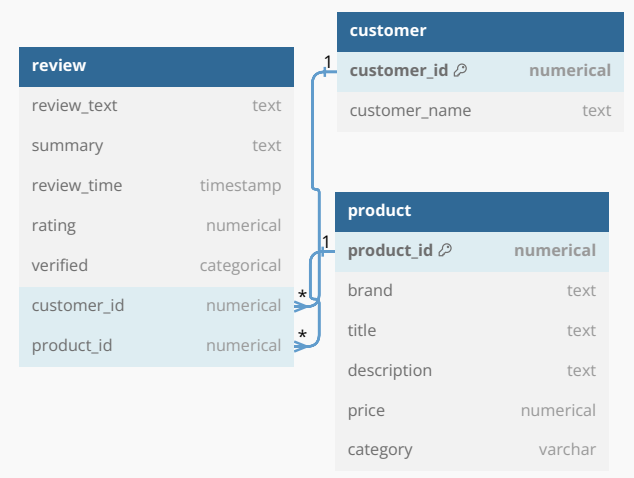

Figure 12: rel-amazon database diagram.

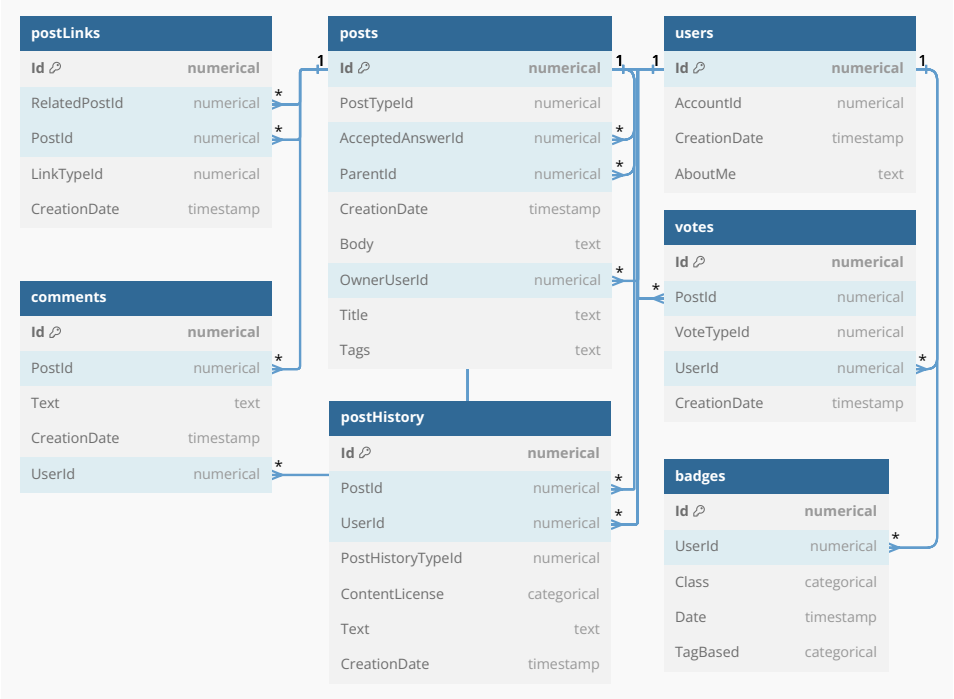

Figure 13: `rel-stack` database diagram.

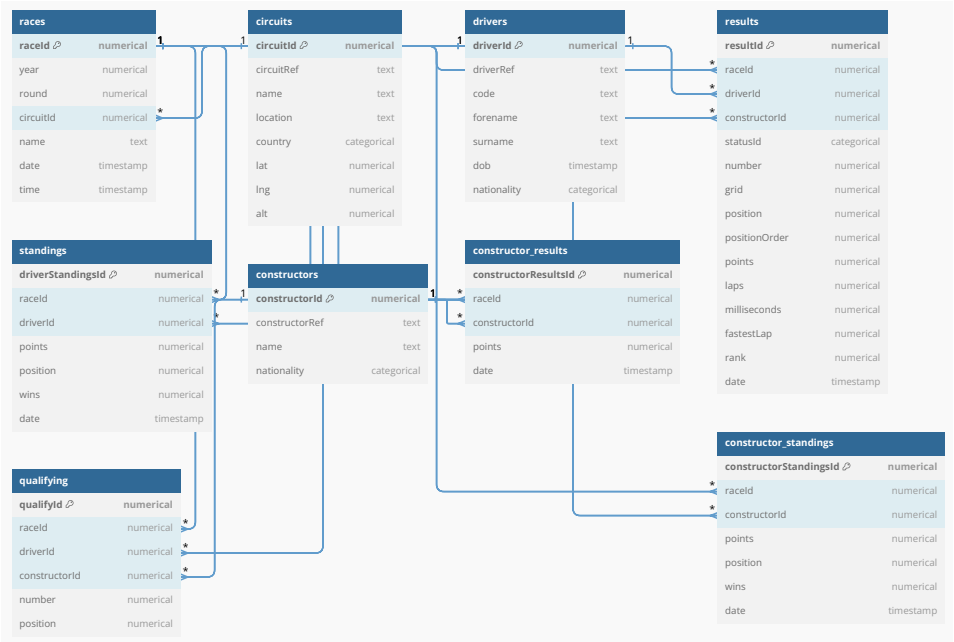

Figure 14: `rel-f1` database diagram.

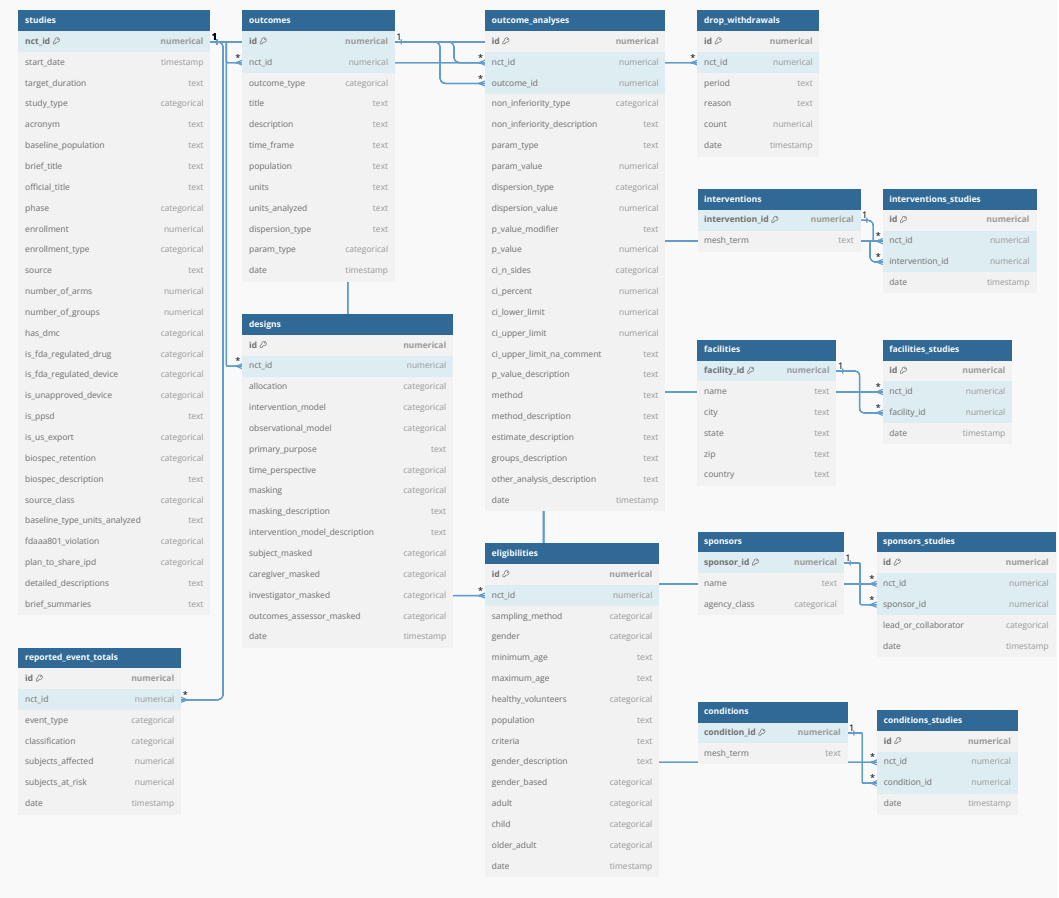

Figure 15: `rel-trial` database diagram.

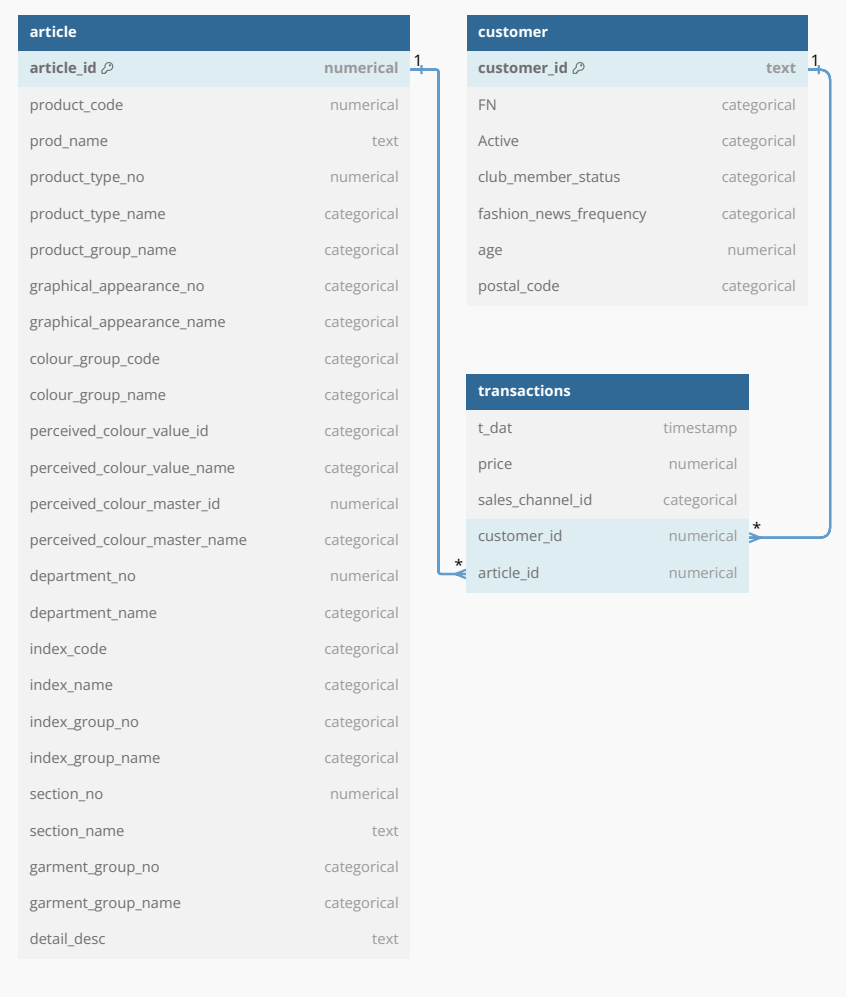

Figure 16: `rel-hm` database diagram.

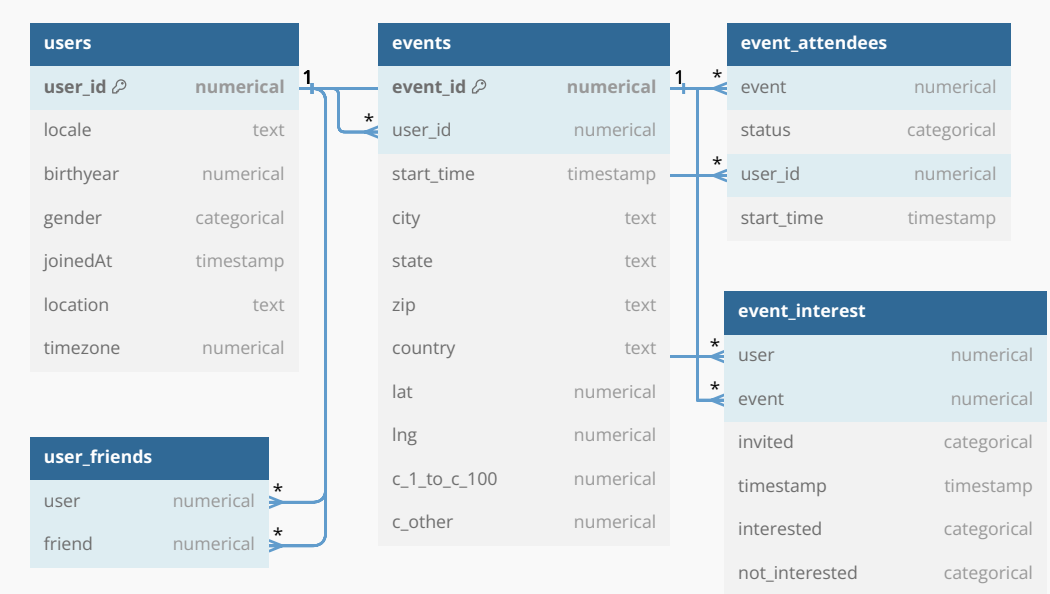

Figure 17: `rel-event` database diagram.

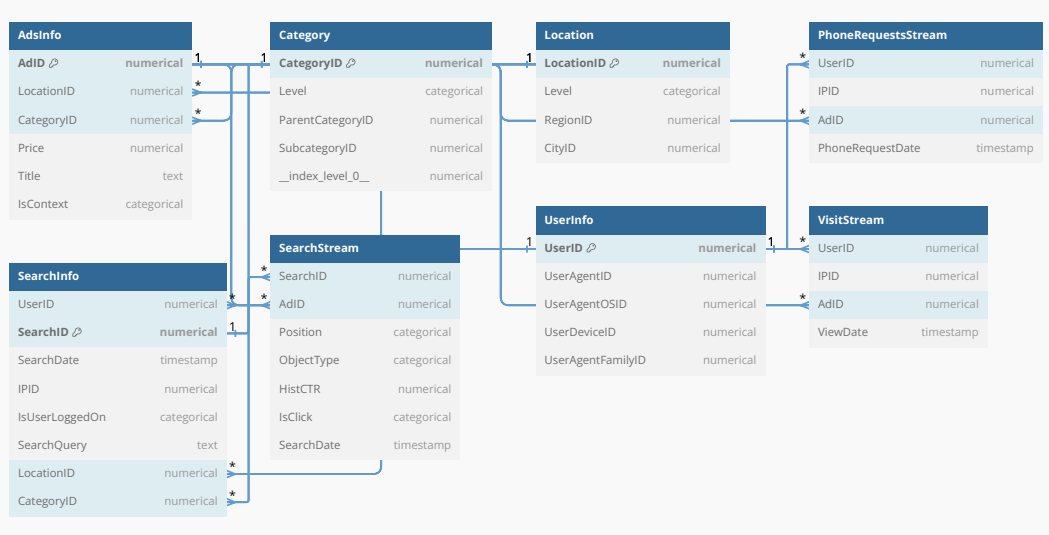

Figure 18: `rel-avito` database diagram.

H Additional Discussion

As well as datasets and model benchmarking, the RELBENCH package aims to make the use of Relational Deep Learning on other problems not considered in this work. Here we discuss several key usability choices make during the design of the RELBENCH package.

Adding new datasets is easy. RELBENCH organizes datasets as subclasses of a parent dataset class. An example definition for `rel-stack` can be found here. This class reads the raw data (e.g., from csv or parquet) into pandas data frame format and stores a dictionary of all the database tables. It also stores a small amount of metadata to indicate which column is the primary key column, which (if any) are foreign-keys linking to another table, and which (if any) column is a timestamp column. All other class functionality is standardized in the parent dataset class, allowing maximal compatibility with the model training and minimal effort to define a new dataset class.

Once defined and added to the database registry, a user can load the data using RELBENCH utility function

```
dataset = get_dataset("dataset-name")
```

and run RELBENCH model training scripts on the data with no further modifications required. Importantly RELBENCH is designed so that this functionality is outward facing—i.e., a user can define their own new dataset and add it to the registry without modifying the package source code. We wrote a tutorial of how to do this here. In practice, we found that writing a new dataset class took no more than a couple of hours.

Data Processing. Models are trained using databases essentially as they are found as this is one of the key promises of Relational Deep Learning. That said, one important pre-processing that was necessary is to drop columns which introduce temporal leakage. This happens when a cell in a row is updated *after* the row is first created, adding information from after the associated timestamp. For example, the raw Stack Exchange database includes a "was answered" TRUE/FALSE column for each question. We deleted this column since providing this as input to any model (RDL or otherwise) would leak information that wouldn't be available at test time. Overwriting of cells is common in real-world relational databases which negatively affects both RDL and data scientist modeling. Designing methods to detect or address this type of temporal leakage would be an interesting direction for future work.

Hardware usage. RELBENCH datasets were chosen to be academic-lab friendly, meaning that all training runs can be run on small GPUs e.g., a 11GB NVIDIA GeForce RTA 2080 Ti released in 2018. We also experimented with Quadro RTX 8000 (48GB) and A100s (80GB). In all cases, loading data to launch an experiment takes no more than 1 minute even for the largest RELBENCH dataset. The one-time text embedding cost is the only significant pre-processing expense. This takes at most 40 minutes for the slowest database, which in this case is the `rel-amazon` database because it has a lot of text data including product descriptions and reviews. We found the computation requirements extremely manageable on modest hardware, and provided the data scientist access to the same hardware for their work. For model training, we found that RDL model training never took more than about 2 hours. In many cases the RDL model trained faster than the data scientist model, primarily because the data scientist model included a comprehensive LightGBM hyperparmeter sweep, whereas the RDL models used a single set of RDL hyperparamers with no tuning.

Reliability of RDL Models. One advantage RDL models forego in comparison to data scientist designed models is the natural interpretability, and associated trust, that comes from features specified through SQL queries. SQL queries often have natural semantic meaning the give insights into possible failure modes of the model.

An intriguing possibility for future work is to view RDL models as compositions of SQL queries, thereby matching the interpretability standards of data scientist-built models. Intuitively the GNN message passing closely resembles a SQL JOIN + AGGREGATE operation. Indeed, since our graph construction connects a node v (i.e., DB entity) to all entities v' with $v_{\text{pkey}} = v'_{\text{fkey}}$, the GNN message passing propagates information from all such v' to v. This is the same as a SQL inner join operation, which makes a new table with a row for all (v, v') pairs, which is often followed by an aggregation (e.g., sum or mode) over v' to get a table with one row for each v. This connection points to a possibility to reverse engineer a set of SQL operations matching each GNN layer.

I 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.