Relational Task Generation Language: A Declarative Specification Framework for Relational Deep Learning

Oleksii Kolesnichenko, Jakub Peleška (✉), and Gustav Šír

Czech Technical University in Prague,
Karlovo náměstí 13, Prague, 121 35, Czechia
kolesole@fel.cvut.cz,{jakub.peleska,gustav.sir}@cvut.cz

Abstract. Relational Deep Learning (RDL) has become a powerful paradigm for learning from multi-tabular data. However, manually defining RDL prediction tasks is a laborious process that frequently results in data leakage. To address this issue, we introduce Relational Task Generation Language (RTGL) - an *open-source* declarative language that streamlines RDL task formulation by abstracting away low-level SQL details. We showcase RTGL by reconstructing existing RDL benchmark tasks and uncovering their inconsistencies stemming from manually crafted SQL definitions of RDL prediction targets, thereby underscoring the value of a dedicated declarative language. In addition, we demonstrate the practical utility of RTGL by designing various new tasks with diverse forms and target types. Our experiments confirm the robustness and usability of RTGL, as well as its seamless integration with the existing RDL frameworks, making it widely accessible to the community.

Keywords: Relational Deep Learning · Domain Specific Language · Predictive Task Generation · Graph Neural Networks · Relational Databases

1 Introduction

Relational databases (RDBs) are ubiquitous for storing and managing structured data across modern enterprises [7]. Yet, leveraging deep learning [14] on this data remains highly challenging [4]. While classic machine learning (ML) domains like computer vision readily represent data in the suitable form of fixed-size numeric tensors [27,20,40,9], relational data, fragmented across multiple interrelated tables of diverse shapes and forms, is fundamentally different. To adapt relational data for classic ML, practitioners used to perform complex joins and aggregations to first denormalize the RDBs [26] into the suitable format. However, this manual feature engineering is not only time-consuming but also inevitably discards information while disrupting the original RDB structure.

To learn from these multi-table structures without the need for manual feature flattening, Relational Deep Learning (RDL) [44,12] has recently emerged as a new ML paradigm. Rather than forcing the data into a fixed-form tensor,

RDL treats the entire database as a temporal heterogeneous graph [38]. In this representation, tables are translated into distinct node types, and the foreign-key relationships between them define the edges. Once this graph is constructed, it can be directly processed by specialized graph neural architectures [34,6,37] designed to capture both the semantic features of individual rows as well as the structural topology of the RDB.

While RDL offers an elegant model architecture blueprint, the practical bottleneck has moved to defining the prediction tasks themselves. Particularly, in relational learning [8], training depends on precisely specified prediction problems—what entity to forecast, which label to target, and the exact timestamp of prediction. Deriving such training data demands complex, time-aware SQL [5] queries, where engineers must enforce strict temporal cutoffs so the model only sees past data, fully isolated from future outcomes [38,33]. This, mostly manual, process is error-prone and often leads to temporal leakage, where information from the future unintentionally contaminates the training features.

Very recently, a *Predictive Query Language* (PQL) [24], was introduced to simplify task generation in RDL by providing a high-level abstraction specifically tailored for predictive modeling on relational data. However, PQL remains a proprietary, closed-source language, deeply embedded within a specific commercial platform,[1] which severely limits its accessibility, transparency, and extensibility for the broader RDL research community.

To address this limitation, we introduce the Relational Task Generation Language (RTGL), an *open-source* declarative language for generating tasks in RDL, which we make publicly available[2] to the community. While RTGL largely adopts the syntax and conceptual goals of the PQL, it is a completely independent project designed to integrate into the wider existing open-source RDL ecosystems, such as RelBench [38] and ReDeLEx [33].

RTGL allows researchers to define complex temporal prediction tasks using but a few lines of declarative code, as depicted in Figure 1. We demonstrate RTGL's capabilities by reconstructing existing RelBench [38] tasks and identifying inconsistencies arising from manually written, complex SQL definitions of RDL prediction targets—highlighting the value of a new dedicated language. We further showcase RTGL by formulating various new tasks within the ReDeLEx framework [33] with diverse target types, emphasizing its practical utility.

2 Related Work

While Tabular Learning methods (e.g., TabNet [1], tabular ResNet [16]) and in-database machine learning extensions (e.g., BigQuery ML [15], PostgresML [36], SQL4ML [30]) have simplified model deployment, they remain fundamentally constrained by their strict requirement for single, flat input tables. To capture the heterogeneity of relational databases, recent advances in RDL have shifted toward integrating tabular representation learning [29,34] with graph

[1] developed by the team at `Kumo.ai`

[2] at `https://github.com/kolesole/RTGL`

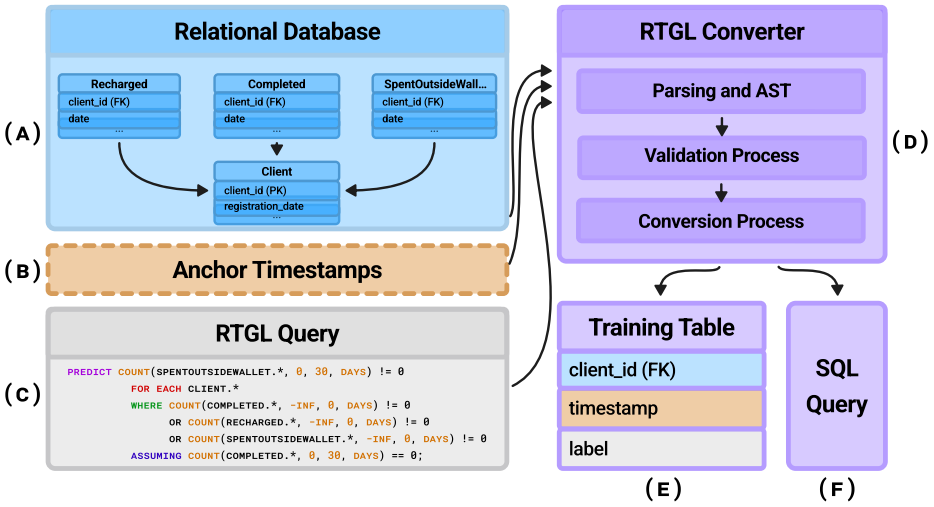

Fig. 1: **RTGL Pipeline.** Schematic overview of the system architecture, detailing the required inputs—(A) a relational database, (C) an RTGL query, and (B) optional anchor timestamps for temporal tasks—(D) the three-stage RTGL converter engine, and the final outputs—(E) a materialized training table or (F) a raw time-safe SQL query.

representation learning [45], large language models [43], and graph transformers [10,28]. This progress is driving the field toward RDL foundation models [42,11,13,22,41,10], supported by task-agnostic pretraining [35] and synthetic data generation [23,25]. To support the architectural evolution, standardized benchmarking ecosystems like RelBench [38,17] and ReDeLEx [33] have formalized predictive tasks with chronological splits to prevent temporal leakage [24] and expanded evaluation across diverse databases to analyze structural impacts.

3 Relational Deep Learning

Relational Deep Learning is a machine learning paradigm built on interpreting relational databases as heterogeneous graphs. This perspective enables direct application of deep learning techniques—specifically graph neural networks alongside tabular representation learning—on the native structures of RDBs. While the underlying idea was explored in earlier work [39,3,2], it was recently formalized under the RDL umbrella [12] to consolidate these efforts into a cohesive research field, detailed below.

3.1 Graph Representation

Formally, a relational database $\mathcal{R}$ is defined as a finite set of relations $R_1, R_2, \ldots, R_n$. Each relation R consists of a heading (signature) $R_{/n}$, formed

by the set of its attributes, and a body, formed by the tuples of attribute values $t_i = (a_1, a_2, \ldots, a_n)$, commonly represented as a table T_R. Additionally, the relation R can be defined extensionally by the unordered set of its tuples: $R = \{t_1, t_2, \ldots, t_m\}$, corresponding to the rows (or entities) of the table T_R.

Practically, the attributes forming the rows of each table T_R are categorized into four functional groups: (i) *primary key*, a minimal set of attributes PK_R that uniquely identifies each entity, such that for any two distinct tuples $t_a, t_b \in R$, $t_a[PK_R] \neq t_b[PK_R]$; (ii) *foreign key*, a set of attributes in a relation R_i that references the primary key of another relation R_j, establishing a structural dependency ensuring referential integrity $\forall t \in R_i, t[FK_{R_i \to R_j}] \in \{t'[PK_{R_j}] \mid t' \in R_j\}$; (iii) *feature attributes*, containing characteristic information about the entity; and, optionally, (iv) a *timestamp*, indicating when the row was created.

The *relational entity graph* of a database $\mathcal{R}$ is a temporal heterogeneous graph defined as $G = (\mathcal{V}, \mathcal{E}, \phi, \psi, \tau)$. The set of nodes $\mathcal{V}$ is defined as the union of all tuples from each relation: $\mathcal{V} = \{v_{i,j} \mid R_i \in \mathcal{R}, t_j \in R_i\}$, inferring $v_{i,j} \simeq t_j \in R_i$. Consequently, the set of edges $\mathcal{E}$ is defined by the primary-key to foreign-key relationships: $\mathcal{E} = \{(v_{i,k}, v_{j,l}) \mid t_k \in R_i, t_l \in R_j, t_k[FK_{R_j}] = t_l[PK_{R_j}] \vee t_l[FK_{R_i}] = t_k[PK_{R_i}]\}$.

To capture the heterogeneity of the database, the function $\phi : \mathcal{V} \to \mathcal{T}^v$ serves as a node mapping. The set of node types $\mathcal{T}^v$ corresponds directly to the set of relations within $\mathcal{R}$, meaning ϕ maps a node $v_{i,j}$ strictly to its original source relation R_i. Similarly, $\psi : \mathcal{E} \to \mathcal{T}^e$ is an edge mapping function, where the set of edge types $\mathcal{T}^e$ corresponds to the specific pairs of key attributes (or their reversed counterparts) that generated the edge. Finally, the time mapping function $\tau : \mathcal{V} \to \mathrm{T}$ assigns a timestamp to each node such that $\tau(v) = \mathrm{T}_v$. If a temporal attribute is not available, a zero timestamp is assigned by default.

3.2 Neural Models

Building upon the graph representation G, RDL models typically follow a four-stage pipeline: (i) a *table-level attribute encoder* initializes node embedding matrices $h_v^{(0)} \in \mathbb{R}^{d^{(0)} \times n}$ by encoding each feature attribute $A_1, \ldots, A_n$ of the node's source relation $\phi(v)$ based on its semantic data type; (ii) a *table-level tabular model* employs tabular architectures to refine these embeddings into $h_v^{(l)} \in \mathbb{R}^{d^{(l)} \times n}$, optionally compressing the matrix into a single vector $h_v^{(l)} \in \mathbb{R}^{d_{\phi(v)}^{(l)}}$; (iii) a *graph neural model* applies message-passing based on the defined edge types $\mathcal{T}^e$, using standard graph neural networks for vectors or custom schemes for attribute-matrices; and (iv) a *task-specific model head* maps the final node embeddings to predictions, typically via simple multi-layer perceptrons.

3.3 Predictive Tasks

Predictive tasks in RDL are formalized by constructing a dedicated *task table* T_{task} that extends the existing relational database $\mathcal{R}$ [38,24]. A task table T_{task} consists of tuples defining the individual prediction instances, abstracted

Fig. 2: **Training Tables.** Illustration of training tables for temporal and static prediction tasks. Both table types contain an `entity id` foreign key that references the target table (e.g., a client id), and a `label` column specifying the prediction target (e.g., number of client orders). The temporal training table additionally includes a `timestamp` column, indicating when the prediction is made.

as (v, y, t). Here, $v \in \mathcal{V}$ is the target entity (node) identified via a foreign key reference $T_{task}[FK]$, $y \in \mathcal{Y}$ is the ground-truth target label to be predicted (e.g., $y \in \mathbb{R}$ for regression), and $t \in \mathrm{T}$ is the anchor time, specifying the moment at which the prediction is made.

The objective of an RDL model f_θ is to predict the target label $\hat{y} = f_\theta(G_{v,\leq t}) \approx y$, where $G_{v,\leq t}$ is the relational subgraph defining the computational context for entity v, strictly filtered to only include information available at or before anchor time t. Depending on the role of t, these tasks are categorized into static and temporal formulations. The general structure of the training tables for both types of tasks is illustrated in Figure 2.

Static Tasks. Static tasks involve predicting missing values or properties within a database viewed as a single, fixed snapshot in time. The temporal dimension is effectively ignored, reducing instances to pairs (v, y). The objective simplifies to learning a function $\hat{y} = f_\theta(G_v)$, utilizing the entire available graph G without the risk of temporal information leakage. In these cases, the target y forms a feature column that already structurally exists within the database but is masked out.

Temporal Tasks. Conversely, temporal tasks (or "forecasting" tasks) predict future behaviors or events that have not yet occurred. Here, the anchor time t is crucial, as the label y represents an aggregation or occurrence in a future window $(t, t + \Delta w]$. The set of task labels is formed as $Y = \{Q_{\Delta w}(t) \mid t \in \mathrm{T}_{task}\}$, where Q is a future-pointing task query with a fixed window size Δw, and T_{task} is a set of task anchor timestamps. To prevent temporal data leakage, the model must strictly condition its representations on past data, requiring the graph to be temporally masked to yield $G_{\leq t} = (\mathcal{V}_{\leq t}, \mathcal{E}_{\leq t})$, where $\mathcal{V}_{\leq t} = \{u \in \mathcal{V} \mid \tau(u) \leq t\}$.

4 Relational Task Generation Language

To automatically construct the training tables (Fig. 2) needed for RDL (Sec. 3.3), we developed RTGL, modeled on the features of the proprietary PQL [24]. The language satisfies several fundamental criteria: it is *declarative*, enabling users to

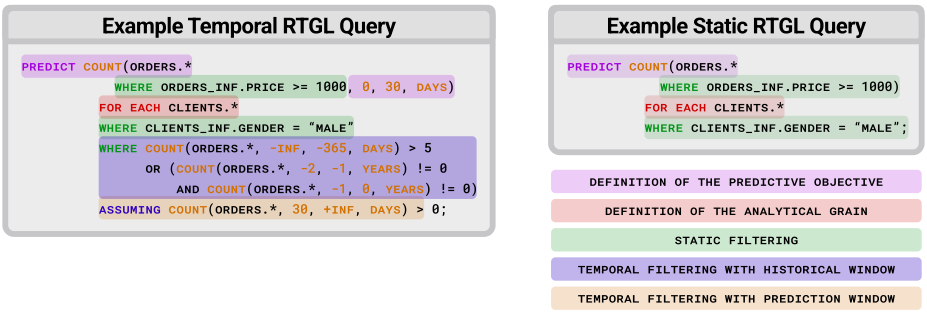

Fig. 3: **RTGL Query Examples.** In both queries, the predictive goal is to determine the total number of orders with a price of at least 1000 placed by male clients. The static query (right) evaluates this condition over the entire global state, whereas the temporal query (left) adds constraints that limit the aggregation to events occurring within the historical observation and prediction window (30 days) defined around the reference timestamp.

state what to predict rather than how to obtain it; it enforces *temporal safety* by clearly distinguishing historical from predictive time windows to avoid data leakage [18]; and it is broadly applicable across *different task types*, including classification, regression, and link prediction. While RTGL adopts its main structure, syntactic patterns, and keywords from PQL,[3] it forms a completely independent, *open-source* project.

4.1 Language Syntax

Following a SQL-like syntax [5], RTGL expresses predictive RDL tasks by specifying what to predict, at which entity, and within what (temporally safe) future and historical time context.

Particularly, the `PREDICT` keyword serves as the entry point for defining the predictive objective of the task, dictating the target label to be computed for each prediction instance (Sec.3.3), as illustrated in the query of Figure 3, where the objective is to predict the `COUNT` of `orders.*`. This clause supports mathematical aggregations, logical conditions, or direct references to raw column values in static scenarios. For temporal queries (Fig. 3), all features evaluated within this block must strictly reference a defined future time window to prevent leakage.

The `FOR EACH` keyword defines the primary entity table (Sec. 3.3) and the specific identifier column that acts as the focal point for the prediction. By binding the extraction logic to a specific node type in the heterogeneous graph (Sec. 3.1), this clause ensures that the resulting task table accurately maps predictions back to the correct entities via their primary keys. In Figure 3, `FOR`

[3] For a more comprehensive account of the language, we recommend that the reader refer to Kumo.ai's official documentation at `https://kumo.ai/docs/predictive-query/`, or the associated publication [24].

`EACH clients` binds the predictions to individual users, establishing the analytical grain of the predictive task.

Aggregation operators, such as `SUM`, `COUNT`, and `AVG`, are fundamental for compressing future or historical records into single scalar values. As summarized in Table 1, RTGL supports a comprehensive set of mathematical and logical operators tailored to specific data types. For temporal tasks, these aggregations explicitly require defined start and end time shifts, forcing the user to map out the chronological bounds of the operation. Combining these aggregations with comparison operators then formulates regression or binary classification tasks.

The `ASSUMING` keyword governs the future prediction window. This clause applies filtering logic directed toward future events relative to the anchor timestamp. For instance, in Figure 3, the temporal query applies `ASSUMING COUNT(orders., 30, +INF, DAY)` to restrict the result to entities that still have at least one order occurring after the 30-day prediction window.

The `WHERE` keyword governs the historical observation window. By looking strictly backward from the anchor timestamp, it bounds the historical context needed for prediction, allowing the framework to select training instances according to prior behaviors or conditions, as illustrated in Figure 3, where the clause `WHERE COUNT(orders., -1, 0, YEARS) =! 0`, limits the training data to include only clients who have placed at least one order in the year preceding the anchor timestamp. Additionally, `WHERE` can be used for static filtering, without temporal constraints, in which case its behavior coincides with that of standard SQL [5].

Finally, the `LIST_DISTINCT` operator is utilized for tasks requiring set-based predictions, such as multilabel classification or link prediction. Unlike standard aggregations that return a scalar value, this operator returns an aggregated set of distinct categorical entities or foreign keys over a specified temporal window. This feature is important for scenarios where the objective is to predict multiple future interactions, such as recommending a list of items a user might purchase.

5 Internal Design and Implementation

The internal implementation of RTGL translates the declarative syntax (Sec. 4) into robust, executable SQL while guaranteeing semantic and chronological correctness. As depicted in the RTGL pipeline (Fig. 1), the converter engine (Fig. 1D) operates on a raw RTGL query (Fig. 1C), a relational database schema (Fig. 1A), and optional anchor timestamps (Fig. 1B). To retain a modular architecture, the engine is conceptually divided into three sequential steps: parsing and AST generation, semantic validation, and query conversion.

5.1 Parsing and AST Generation

The first operational step transforms the raw query string (Fig. 1C) into a structured format. This phase is implemented with the use of ANTLR4 [32] parser generator, chosen for its robust adaptive LL(*) parsing algorithm and its ability

Type	Aggregation	Condition
Numerical	`AVG`, `COUNT`, `COUNT_DISTINCT`, `FIRST`, `LAST`, `MAX`, `MIN`, `SUM`	`!=`, `<`, `<=`, `==`, `>`, `>=`
String	`FIRST`, `LAST`	`NOT LIKE`, `NOT CONTAINS`, `ENDS WITH`, `STARTS WITH`, `LIKE`, `CONTAINS`, `==`
Nullable	`AVG`, `COUNT`, `COUNT_DISTINCT`, `FIRST`, `LAST`, `MAX`, `MIN`, `SUM`, `LIST_DISTINCT`	`IS NULL`, `IS NOT NULL`

Table 1: **Aggregation and Condition Functions in RTGL.** The exhaustive list of aggregations and conditions available in RTGL for each data type, mapping closely to the foundational capabilities outlined in PQL.

to automatically generate a Concrete Syntax Tree (CST) based on the formal grammar. However, the raw CST contains redundant syntactical artifacts. To extract meaningful semantic logic, the engine employs a Visitor traversal pattern. This process selectively explores the tree and dynamically constructs an Abstract Syntax Tree (AST), standardizing the representation into a nested dictionary. This structure effectively isolates the raw text parsing from the deeper logical operations while preserving positional metadata (line and column numbers) necessary for precise error reporting.

5.2 Semantic Validation

While ANTLR4 enforces syntactic correctness, a grammatically valid query is not necessarily semantically sound or executable within the context of the given database. The generated AST is thus passed to the validation process, which ensures logical correctness against the provided relational schema (Fig. 1A). The validation module verifies the existence of all referenced tables and columns, and ensures that primary and foreign key constraints are valid. Notably, the process performs strict type-checking, verifying that the data types of the columns align perfectly with the conditional operators applied to them. Furthermore, temporal validation enforces strict chronological logic: observation windows must reference negative time shifts (the past), whereas predictive windows must reference positive time shifts (the future). A key architectural principle of this module is its avoidance of a *fail-fast* paradigm. Rather than halting at the first detected error, the validation process accumulates all syntactic and semantic violations across the entire AST to provide a comprehensive error report to the user before any data extraction occurs.

rel-f1 driver-top3

```
PREDICT MIN(QUALIFYING.POSITION, 0, 30, DAYS) <= 3
        FOR EACH DRIVERS.*
        ASSUMING COUNT(PROBEHNUTO.*, 0, 30, DAYS) == 0;
```

(a) **rel-f1 driver-top3.** Binary classification task, for each driver predict if they will qualify in the top-3 for a race in the next 1 month.

rel-f1 driver-position

```
PREDICT AVG(RESULTS.POSITIONORDER, 0, 60, DAYS) != 1
        FOR EACH DRIVERS.*;
```

(b) **rel-f1 driver-position.** Regression task, predict the average finishing position of each driver in the next 2 months.

rel-f1 driver-dnf

```
PREDICT MAX(RESULTS.STATUSID, 0, 30, DAYS) != 1
        FOR EACH DRIVERS.*
        WHERE COUNT(RESULTS.*, -365, 0, DAYS) != 0
        ASSUMING MAX(RESULTS.STATUSID, 0, 30, DAYS) IS NOT NULL;
```

(c) **rel-f1 driver-dnf.** Binary classification task, for each driver predict not finishing a race (DNF) in the next 1 month.

Fig. 4: Recreated rel-f1 dataset tasks from RelBench [17] benchmark.

5.3 Query Conversion

Once the query passes all semantic checks, the AST is processed by the conversion module to materialize the executable logic. Instead of generating a single, massive SQL statement, the converter dynamically constructs isolated relational algebra subtables for distinct logical expressions by recursively unwrapping the AST dictionary. These subtables are then combined using formal relational operations; for instance, boolean `AND` conditions translate into SQL `INTERSECT` operations, while `OR` conditions translate into `UNION` operations. For temporal tasks, the conversion engine leverages Common Table Expressions (CTEs) [31] to inject the required anchor timestamps (Fig. 1B) directly at the top of the SQL query. All subsequent intermediate subtables and historical observation windows are strictly computed relative to these anchor timestamps, preventing temporal leakage at the execution level. The final output is returned either as a materialized training table (Fig. 1E) or as a raw, time-safe SQL query (Fig. 1F).

6 Validation and Experiments

The primary objective of our experiments is to demonstrate that RDL task generation via RTGL is simple, correct, and practically viable. To validate the framework, the experimental pipeline is divided into empirical baseline verification against the established RelBench benchmark [38], an extensibility demonstration using the ReDeLEx framework [33], and a structural integrity confirmation via end-to-end model training. Importantly, all tasks, along with the source code for the experiments that utilize RTGL, are freely accessible on GitHub.[4]

[4] `https://github.com/kolesole/rtgl-tasks`

6.1 Recreated Tasks

To establish an empirical baseline and verify the correctness of RTGL, we regenerated several pre-defined tasks from the RelBench benchmark across the *rel-f1* (Fig. 4) and *rel-stack* (Appendix A.2, Fig. 6) datasets. The majority of standard predictive tasks were successfully recreated using simple declarative statements.

To empirically demonstrate the consistency of the generated relational tables, we joined them with the RelBench reference tables to identify any discrepancies. This direct comparison guarantees equivalence between the RTGL outputs and the established baselines for correctly formulated tasks. We note that special tasks requiring deeply nested, multi-hop connections without direct foreign key paths could not be natively modeled in the current version of RTGL without defining intermediate database views. As of current, RTGL prioritizes structural transparency by explicitly defining each table used within the task definition.

6.2 Found Inconsistencies

Importantly, the process of re-defining these tasks using our declarative engine yielded an unexpected but essential outcome: it revealed that the official RelBench benchmark[5] contains underlying logical and structural errors within its manually constructed SQL tasks. By comparing the tables, we identified problems where the original RelBench tables incorrectly contained more rows than the temporally sound RTGL tables.

For instance, in the *driver-dnf* (Fig. 4c) binary classification task, analyzing the original manual SQL query revealed a error in its temporal logic. The query filtered for active drivers but failed to apply an upper-bound constraint relative to the current timestamp. Consequently, the query looked into the future, filtering the main table based on driver activity that had not yet occurred at the prediction moment, and occasionally included drivers that did not yet exist in the database.

A similar mismatch occurred during the validation of the *user-engagement* and *user-post-comment* tasks. In these tasks, the original RelBench SQL failed to filter users by their creation date. As a result, the baseline tables included records for users who had not yet registered at the given prediction timestamps. This demonstrates that manually writing temporal SQL queries is a highly error-prone process, with RTGL's strict bounding automatically preventing such scenarios.

6.3 New Tasks

Following the verification of the framework against established benchmarks, we demonstrate the practical extensibility of RTGL by introducing novel predictive tasks, as shown in Figure 5. These new tasks were formulated for the real-world *Seznam* dataset from the ReDeLEx framework [33], which is a highly complex, enterprise-scale relational database.[6] RTGL successfully parsed complex

[5] `https://relbench.stanford.edu`
[6] derived from the Czech web portal `seznam.cz`

ctu-seznam client-outside

```
PREDICT COUNT(PROBEHNUTO_MIMO_PENEZENKU.*, 0, 30, DAYS) != 0
        FOR EACH CLIENT.*
        WHERE COUNT(PROBEHNUTO.*, -INF, 0, DAYS) != 0
              OR COUNT(DOBITO.*, -INF, 0, DAYS) != 0
              OR COUNT(PROBEHNUTO_MIMO_PENEZENKU.*, -INF, 0, DAYS) != 0
        ASSUMING COUNT(PROBEHNUTO.*, 0, 30, DAYS) == 0;
```

(a) **ctu-seznam client-outside.** Binary classification task, predict whether a client will spend money outside wallet in the next 30 days.

ctu-seznam client-first-service

```
PREDICT FIRST(PROBEHNUTO.SLUZBA, 0, 30, DAYS)
        FOR EACH CLIENT.*
        WHERE COUNT(PROBEHNUTO.*, -INF, 0, DAYS) != 0
              OR COUNT(DOBITO.*, -INF, 0, DAYS) != 0
              OR COUNT(PROBEHNUTO_MIMO_PENEZENKU.*, -INF, 0, DAYS) != 0;
```

(b) **ctu-seznam client-first-service.** Multiclass classification task, predict the first service a client will use in the next 30 days.

ctu-seznam client-services

```
PREDICT LIST_DISTINCT(PROBEHNUTO.SLUZBA, 0, 30, DAYS)
        FOR EACH CLIENT.*
        WHERE COUNT(PROBEHNUTO.*, -INF, 0, DAYS) != 0
              OR COUNT(DOBITO.*, -INF, 0, DAYS) != 0
              OR COUNT(PROBEHNUTO_MIMO_PENEZENKU.*, -INF, 0, DAYS) != 0;
```

(c) **ctu-seznam client-services.** Multilabel classification task, predict services client will use in the next 30 days.

ctu-seznam client-spending

```
PREDICT SUM(PROBEHNUTO.KC_PROKLIKANO, 0, 30, DAYS)
        FOR EACH CLIENT.*
        WHERE COUNT(PROBEHNUTO.*, -INF, 0, DAYS) != 0
              OR COUNT(DOBITO.*, -INF, 0, DAYS) != 0
              OR COUNT(PROBEHNUTO_MIMO_PENEZENKU.*, -INF, 0, DAYS) != 0;
```

(d) **ctu-seznam client-spending.** Binary classification task, predict client spending amount in the next 30 days.

Fig. 5: Novel tasks on ctu-seznam dataset from ReDeLEx [33] framework.

schema relationships to define a variety of novel tasks including binary classification, multiclass classification, multilabel classification, and regression. This demonstrates the high-level abstraction and simplicity of using the framework to generalize beyond curated academic benchmarks.

6.4 Results

To confirm the structural integrity and practical utility of the generated tasks, we selected several re-defined tasks from RelBench, alongside the newly defined ReDeLEx tasks, and trained predictive RDL models (Sec. 3.2) on them. We evaluated two structurally different neural architectures configured with comparable complexity: a GraphSAGE baseline utilizing a mean aggregator, and a Heterogeneous Graph Transformer (HGT) utilizing two attention heads, representing a more advanced RDL approach. Detailed hyperparameter configurations are provided in Appendix A.1.

The results are presented in Table 2 in the form of the mean and standard deviation of the test and validation metrics. Both models showed strong predictive capabilities and consistent improvement during training across all diverse task types. The successful convergence confirms that the task tables generated by the RTGL engine are structurally sound and temporally valid. Notably, the simpler GraphSAGE baseline frequently surpassed the HGT model, in line with recent evidence in relational deep learning [10].

Dataset	Task	Split	RDL Model	
			SAGE	HGT
Binary Classification (Accuracy ↑)				
rel-f1	driver-dnf	val	.8026 ± .0052	.7996 ± .0089
		test	.7219 ±.0123	.6932 ± .0138
ctu-seznam	client-outside	val	.9764 ± .0016	.9868 ± 0
		test	.9768 ± .0016	.9876 ±0
Multiclass Classification (Accuracy ↑)				
ctu-seznam	client-first-service	val	.4417 ± .0058	.4364 ± .0036
		test	.4677 ±.0056	.4585 ± .0005
Regression (MAE ↓)				
rel-f1	driver-position	val	2.9798 ± .0803	3.8292 ± .0154
		test	3.9138 ±.0819	4.2645 ± .0597
ctu-seznam	client-spending	val	8826.14 ± 1.11	8839.10 ± 3.06
		test	9306.87 ±1.54	9321.03 ± 2.88
Multilabel Classification (AUPRC ↑)				
ctu-seznam	client-services	val	.6217 ± .0017	.5929 ± .0074
		test	.6274 ±.0017	.5987 ± .0078

Table 2: **RDL model performance on selection of RTGL defined tasks.** Metrics vary depending on the objective: Accuracy for binary and multiclass classification, Mean Absolute Error (MAE) for regression, and Area Under the Precision-Recall Curve (AUPRC) for multilabel classification.

7 Conclusion

This work addresses the challenge of predictive task generation in relational deep learning. We showed that manual creation of temporal predictive tasks is a laborious process that frequently results in data leakage. We presented RTGL, an open-source declarative language that simplifies predictive task formulation by abstracting away low-level SQL logic. Our experiments demonstrated that using RTGL to generate standard benchmarks uncovered temporal inconsistencies in official manual definitions. Furthermore, RTGL generated valid structural data for novel tasks, enabling the successful training of downstream RDL models.

While RTGL provides a solid foundation, several limitations remain for future work. Particularly, expanding the syntax to natively support interactions between distant tables separated by multi-hop connections, without requiring intermediate views, will further enhance the framework's expressiveness. Additionally, performance optimization is necessary to accelerate data processing for massive enterprise datasets.

References

1. Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, pages 6679–6687, 2021.
2. Vojtěch Aschenbrenner. Deep relational learning with predicate invention. M.Sc. thesis, Czech Technical University in Prague, 2013.
3. H Blockeel and W Uwents. Using neural networks for relational learning. In *ICML-2004 Workshop on Statistical Relational Learning and its Connection to Other Fields*, 2004.
4. Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE transactions on neural networks and learning systems*, 2022.
5. Donald D. Chamberlin and Raymond F. Boyce. SEQUEL: A structured English query language. In *Proceedings of the 1974 ACM SIGFIDET (now SIGMOD) workshop on Data description, access and control*, SIGFIDET '74, pages 249–264, New York, NY, USA, 1974. Association for Computing Machinery.
6. Tianlang Chen, Charilaos Kanatsoulis, and Jure Leskovec. RelGNN: Composite Message Passing for Relational Deep Learning. *arXiv preprint arXiv:2502.06784*, 2025.
7. Edgar F Codd. *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., 1990.
8. Luc De Raedt. *Logical and Relational Learning*. Springer, 2008.
9. Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pages 4171–4186, 2019.
10. Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico Lopez, Charilaos I. Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational graph transformer. In *The Fourteenth International Conference on Learning Representations*, 2026.
11. Vijay Prakash Dwivedi, Charilaos Kanatsoulis, Shenyang Huang, and Jure Leskovec. Relational deep learning: Challenges, foundations and next-generation architectures. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '25, page 5999–6009, New York, NY, USA, 2025. Association for Computing Machinery.
12. Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Position: Relational deep learning - graph representation learning on relational databases. In *Forty-first International Conference on Machine Learning*, 2024.
13. Matthias Fey, Vid Kocijan, Federico Lopez, Jan Eric Lenssen, and Jure Leskovec. KumoRFM: A Foundation Model for In-Context Learning on Relational Data. May 2025.
14. Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
15. Google Cloud. *BigQuery ML Documentation*. Google LLC, 2026. Accessed: 2026-05-11.
16. Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In *Proceedings of the 35th International*

Conference on Neural Information Processing Systems, NIPS '21, pages 18932–18943, Red Hook, NY, USA, 2021. Curran Associates Inc.
17. Justin Gu, Rishabh Ranjan, Charilaos Kanatsoulis, Haiming Tang, Martin Jurkovic, Valter Hudovernik, Mark Znidar, Pranshu Chaturvedi, Parth Shroff, Fengyu Li, and Jure Leskovec. RelBench v2: A Large-Scale Benchmark and Repository for Relational Data, February 2026. arXiv:2602.12606 [cs].
18. Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.
19. Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034, 2017.
20. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.
21. Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. Heterogeneous graph transformer, 2020.
22. Valter Hudovernik, Federico López, Vid Kocijan, Akihiro Nitta, Jan Eric Lenssen, Jure Leskovec, and Matthias Fey. KumoRFM-2: Scaling Foundation Models for Relational Learning, April 2026. arXiv:2604.12596 [cs].
23. Martin Jurkovic, Valter Hudovernik, and Erik Štrumbelj. Syntherela: A benchmark for synthetic relational database generation. In *ICLR Workshop SynthData*, 2025.
24. Vid Kocijan, Jinu Sunil, Jan Eric Lenssen, Viman Deb, Xinwei Xe, Federico Reyes Gomez, Matthias Fey, and Jure Leskovec. Predictive Query Language: A Domain-Specific Language for Predictive Modeling on Relational Databases, February 2026. arXiv:2602.09572 [cs].
25. Vignesh Kothapalli, Rishabh Ranjan, Valter Hudovernik, Vijay Prakash Dwivedi, Johannes Hoffart, Carlos Guestrin, and Jure Leskovec. PluRel: Synthetic Data unlocks Scaling Laws for Relational Foundation Models, February 2026. arXiv:2602.04029 [cs].
26. Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. *Relational data mining*, pages 262–291, 2001.
27. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.
28. Divyansha Lachi, Mahmoud Mohammadi, Joe Meyer, Vinam Arora, Tom Palczewski, and Eva L. Dyer. RGP: A Cross-Attention based Graph Transformer for Relational Deep Learning. October 2025.
29. Veronica Lachi, Antonio Longa, Beatrice Bevilacqua, Bruno Lepri, Andrea Passerini, and Bruno Ribeiro. Boosting Relational Deep Learning with Pretrained Tabular Models, April 2025. arXiv:2504.04934 [cs].
30. Nantia Makrynioti, Ruy Ley-Wild, and Vasilis Vassalos. sql4ml: A declarative end-to-end workflow for machine learning, 2019.
31. Jim Melton. *Advanced SQL: 1999: Understanding object-relational and other advanced features*. Elsevier, 2002.
32. Terence Parr. *The Definitive ANTLR 4 Reference*. Pragmatic Bookshelf, 2nd edition, 2013.
33. Jakub Peleška and Gustav Šír. ReDeLEx: A Framework for Relational Deep Learning Exploration. In Rita P. Ribeiro, Bernhard Pfahringer, Nathalie Japkowicz, Pedro Larrañaga, Alípio M. Jorge, Carlos Soares, Pedro H. Abreu, and João Gama,

editors, *Machine Learning and Knowledge Discovery in Databases. Research Track*, pages 438–456, Cham, September 2025. Springer Nature Switzerland.
34. Jakub Peleška and Gustav Šír. Tabular Transformers Meet Relational Databases. *ACM Trans. Intell. Syst. Technol.*, 16(5):115:1–115:24, September 2025.
35. Jakub Peleška and Gustav Šír. Task-Agnostic Contrastive Pretraining for Relational Deep Learning. In Irena Koprinska, João Mendes-Moreira, and Paula Branco, editors, *Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, pages 623–638, Cham, 2026. Springer Nature Switzerland.
36. PostgresML Team. PostgresML: Postgres with GPUs for machine learning and AI applications, 2026. Accessed: 2026-05-11.
37. Rishabh Ranjan, Valter Hudovernik, Mark Znidar, Charilaos Kanatsoulis, Roshan Upendra, Mahmoud Mohammadi, Joe Meyer, Tom Palczewski, Carlos Guestrin, and Jure Leskovec. Relational Transformer: Toward Zero-Shot Foundation Models for Relational Data, October 2025. arXiv:2510.06377 [cs].
38. Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan Eric Lenssen, Yiwen Yuan, Zecheng Zhang, Xinwei He, and Jure Leskovec. Relbench: A benchmark for deep learning on relational databases. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024.
39. Gustav Šír. *Deep Learning with Relational Logic Representations*. Czech Technical University, 2021.
40. Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 2017.
41. Yanbo Wang, Xiyuan Wang, Quan Gan, Minjie Wang, Qibin Yang, David Wipf, and Muhan Zhang. Griffin: Towards a Graph-Centric Relational Database Foundation Model, May 2025. arXiv:2505.05568 [cs].
42. Johannes Wehrstein, Carsten Binnig, Fatma Özcan, and Shobha Vasudevan. Towards Foundation Database Models. May 2025.
43. Fang Wu, Vijay Prakash Dwivedi, and Jure Leskovec. Large Language Models are Good Relational Learners, June 2025. arXiv:2506.05725 [cs].
44. Lukáš Zahradník, Jan Neumann, and Gustav Šír. A deep learning blueprint for relational databases. In *NeurIPS 2023 Second Table Representation Learning Workshop*, 2023.
45. Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

A Additional Results and Technical Details

A.1 Experiment Details

To enable a fair comparison between models, we tuned hyperparameters to yield similar architectural complexity. Both GraphSAGE [19] and HGT [21] employed a hidden representation of 128 channels. After the graph message-passing layers, node embeddings were passed through a shared MLP head with 256 hidden units, ReLU nonlinearity, layer normalization, and dropout. All experiments used the AdamW optimizer with a learning rate of 0.0005 and a batch size of 256. Training was performed for up to 30 epochs with an early stopping patience of 10. For RTGL tasks, neighborhood sizes per layer were set to 15, 10, and 5, while for RelBench tasks a constant neighborhood size of 5 was applied to all layers to speed up training.

A.2 Additional RTGL Tasks

In addition to *rel-f1* tasks (Fig. 4), we re-created tasks from *rel-stack* dataset (Fig. 6), also from the RelBench [17] benchmark.

rel-stack user-engagement

```
PREDICT COUNT(VOTES.*, 0, 91, DAYS) != 0
     OR COUNT(POSTS.*, 0, 91, DAYS) != 0
     OR COUNT(COMMENTS.*, 0, 91, DAYS) != 0
          FOR EACH CLIENT.*
          ASSUMING COUNT(VOTES.*, -INF, 0, DAYS) != 0
                OR COUNT(POSTS.*, -INF, 0, DAYS) != 0
                OR COUNT(COMMENTS.*, -INF, 0, DAYS) != 0;
```

(a) **rel-stack user-engagement.** Binary classification task, for each user predict if a user will make any votes, posts, or comments in the next 3 months.

rel-stack post-post-related

```
PREDICT LIST_DISTINCT(POSTLINKS.RELATEDPOSTID, 0, 91, DAYS)
         FOR EACH POSTS.*;
```

(b) **rel-stack post-post-related.** Link prediction task, predict a list of existing posts that users will link a given post to in the next two months.

rel-stack user-post-comment

```
PREDICT LIST_DISTINCT(COMMENTS.POSTID
            WHERE POSTS.OWNERUSERID IS NOT NULL
              AND POSTS.OWNERUSERID != -1, 0, 91, DAYS)
         FOR EACH USERS.*;
```

(c) **rel-stack user-post-comment.** Link prediction task, predict a list of existing posts that a user will comment on in the next two months.

Fig. 6: Recreated rel-stack dataset tasks from RelBench benchmark.