Transformers Meet Relational Databases

Jakub Peleška, Gustav Šír

***Abstract*—Transformer models have continuously expanded into all machine learning domains convertible to the underlying sequence-to-sequence representation, including tabular data. However, while ubiquitous, this representation restricts their extension to the more general case of *relational databases*. In this paper, we introduce a modular neural message-passing scheme that closely adheres to the formal relational model, enabling *direct* end-to-end learning of tabular Transformers from database storage systems. We address the challenges of appropriate learning data representation and loading, which are critical in the database setting, and compare our approach against a number of representative models from various related fields across a significantly wide range of datasets. Our results demonstrate a superior performance of this newly proposed class of neural architectures.**

I. Introduction

WHILE the approaches to mathematical modeling of complex systems, ranging from control theory to machine learning (ML), evolved in various independent ways, one aspect remained almost universal — the *data representation*. Irrespective of the used models, from decision trees to neural networks, virtually all ML libraries expect input samples in the form of fixed-size numeric tensors, most often just (feature) vectors. Assuming the data samples as independent points in n-dimensional spaces is extremely convenient and allows for building directly upon the elegant foundations of linear algebra and multivariate statistics [1]. However, actual real-world data is not stored in numeric vectors or tensors but mostly in the interlinked structures of internet pages, knowledge graphs, and, particularly, *relational databases*. Indeed, while there are numerous data storage formats, the traditional relational database management systems (RDBMS) arguably dominate the industry, from medicine and engineering to enterprise application domains [2].

In recent years, we have witnessed *deep learning* to quickly dominate all perceptual domains, from vision and speech to language. Nevertheless, it remains very rare to encounter neural models on the classic *tabular data* with heterogeneous features, where standard statistical models, mainly various decision tree ensembles [3], still appear to lead the benchmarks [4]. Improving the performance of the neural models, primarily the omnipresent *Transformer* architecture [5], on tabular datasets gains increasing amounts of attention, sometimes quoted as the "last unconquered castle" for deep learning [6]. Nevertheless, generalizing Transformers from the tabular to the full *relational* data model posits arguably an even bigger challenge.

Jakub Peleška and Gustav Šír are with the Department of Computer Science, Czech Technical University in Prague, 12135 Prague, Czechia (email: jakub.peleska@fel.cvut.cz, gustav.sir@cvut.cz)

In this paper, we introduce a new class of such deep learning architectures aimed directly at relational database representation while utilizing insights from the established field of *relational learning* [7], which is concerned with such generalizations of statistical models.

The core contribution of our work, put into context of related work in Sec. II-F, is the design of a new neural message-passing scheme following the formal relational model while deeply integrating the existing (tabular) Transformer architectures. The implementation of the proposed framework is readily available at Github. [1]

II. Related Work

While the body of work on using deep learning with relational databases themselves is extremely scarce, there are established machine learning areas that either use neural models on simpler data structures or address relational structures with other (non-neural) models. In this section, we first briefly review these fields, often overlooked in deep learning, to properly position the contribution of our work (Sec. II-F).

A. Tabular models

Tabular neural models [8] are concerned with transferring deep learning strategies into the (classic) tabular data setting, currently still largely dominated by standard statistical models, such as gradient-boosted trees [3]. These commonly aim to amend the Transformer architecture [5] to better fit the complex, often heterogeneous and discrete, attribute structure of the tabular data. Some notable models in this category include the TabNet [9], which uses a custom-modified transformer-based architecture; TabTransfomer [10], which focuses on categorical values while utilizing the original Transformer Encoder structure; SAINT [11], which introduced the concept of inter-sample attention; and Trompt [12], which takes inspiration from prompt learning of language models. We note that these tabular Transformers are sometimes (confusingly) referred to as "relational." However, they do not follow the actual relational (database) model and cannot be (directly) used as such.

B. Statistical relational learning

For decades [13], proper learning with actual relational representations has been the concern of the little-known field of Relational machine learning [14]. It builds heavily on the formalism of first-order logic (FOL) [15], in which the tabular representation and the corresponding models are effectively viewed as *propositional*, while the database representation, corresponding formally to a subset of FOL, requires *relational*

[1] https://github.com/jakubpeleska/deep-db-learning

generalization(s) of such models. Many such FOL-based methods have been proposed, mostly following the paradigm of Inductive Logic Programming (ILP) [16], later extended with probabilistic methods under the umbrella of Statistical Relational Learning (SRL) [7]. The most appropriate SRL works capable of learning from database representations then follow the paradigm of "lifting," [17] referring to the generalization of classic statistical models into the relational setting. However, building on the FOL foundations, the SRL models typically do not scale well and, importantly, do not offer the latent representation learning capabilities of neural networks.

C. Propositionalization

From the SRL view, the Tabular Transformers address the exact same representation expressiveness as their classic tree-based counterparts they aim to surpass. The tabular, also known as "attribute-value," data format is an established ML representation perpetuating the whole field. While much of the real-world data structures, such as relational databases, do *not* fit into this representation, a natural urge arises to transform such structures into the expected format and proceed with the standard models. This practice, generally referred to as *propositionalization* [18], is the traditional method of choice that has dominated the industry [19], [20]. Propositionalization is essentially a data preprocessing routine where relational substructures get extracted and aggregated into standard statistical (tabular) attributes corresponding to various select-join-aggregate (SQL) routines in the database setting. Building on decades of practice, the resulting (statistical) models using the resulting attribute vectors typically perform very well. However, their representation learning capabilities are principally limited, as the preprocessing (denormalization) step *necessarily* introduces an information loss.

D. Neuro-symbolic models

An interesting area on the intersection of proper relational (logical) representations and deep learning is known as Neural-Symbolic Integration [21]. There is a (small) number of neuro-symbolic frameworks that operate with some (subset of) FOL representation, effectively covering the relational databases while marrying the principles of neural networks through deep integration, such as Neural Theorem Provers [22], Logic Tensor Networks [23], or Lifted Relational Neural Networks [24]. These methods are, in theory, capable of actual deep learning from relational databases. However, to the best of our knowledge, none of these methods scales to real-world database sizes due to the complexity associated with their FOL-based foundations, except for those that follow some form of the propositionalization scheme under the hood, such as [25].

E. Deep relational models

The closest related work consists of extending standard neural models towards relational representations. The most prominent models in this category are Graph Neural Networks (GNNs) [26] designed for end-to-end learning with graph-structured data. There are currently hundreds of the original GNN model [27] variants, some of which are close in spirit to our proposal, particularly some of the hyper-graph [28] and multi-relational [29] extensions towards knowledge-graph applications [30]. Nevertheless, the graph-based view adopted within this stream of research is generally not concerned with the salient features specific to relational *databases*, particularly with the rich inner structure of the individual records.

There have been only very few works that address (some of) the database-specific aspects. Particularly, the original work of [31] followed by an (unsuccessful) pre-training procedure in [32], and the work of [33], which further incorporated feature engineering and random architecture search to improve its performance. A different line of work has been to utilize techniques from pre-training (large) language models while treating related database tuples as sentences, similarly to the tabular models [34], such as in [35].

In a similar spirit, the authors of [36] presented a (draft) vision for foundational database models, later shifting focus to scaling up GNNs for the task in [37] by leveraging symmetries. Likewise, a recent position paper of [38] aimed to establish "relational deep learning" as a new machine learning subfield while introducing a framework for benchmarking the GNN models,[2] such as [39], [40], and [41].

F. Our Contributions

Our work can be seen as a continuation of these deep relational learning efforts, most notably the work of [41] that this paper directly expands. Particularly, we extend the existing GNN paradigm by tightly integrating the Transformer architecture into the relational message-passing scheme. Thus, apart from proper treatment of the inter-relational structure, we also incorporate, in the spirit of the tabular Transformers, the *intra-relational* structure of the attributes, embedded end-to-end within the same learning scheme. Covering the GNN efforts as a special case, we introduce the most complete framework for deep learning with *actual* relational (SQL) databases, demonstrating superior results over the widest range of available benchmark datasets reported thus far.

III. Background

A. Relational Databases

The principles of relational databases are formally based on the *relational model* [42], rooted in FOL [15], providing a unified declarative specification for managing structured data, irrespective of the particular software implementation. This abstraction allows the definition of any database as a collection of n-ary relations defined over the domains of their respective attributes, managed by the RDBM system to ensure consistency of the data with the integrity constraints of the logical database schema. The key concepts to be used in this paper are as follows.

[2] focusing heavily on the temporal dimension of database records, which we explore experimentally in App. C

1) Relation (Table): Formally, an n-ary relation $R_{/n}$ is a subset of the Cartesian product defined over the domains D_i of its n *attributes* A_i as $R_{/n} \subseteq D_1 \times D_2 \times \cdots \times D_n$, where $D_i = \mathsf{dom}(A_i)$. Each relation R consists of a heading (signature) $R_{/n}$, formed by the set of its attributes, and a body, formed by the particular attribute values, which is commonly viewed as a *table* T_R of the relation R.

2) Attribute (Column): *Attributes* $\mathcal{A}_\mathcal{R} = \{A_1, \ldots, A_n\}$ define the terms of a relation $R_{/n}$, corresponding to the *columns* of the respective table T_R. Each attribute is a pair of the attribute's name and a *type*, constraining the domain of each attribute as $\mathsf{dom}(A_i) \subseteq \mathsf{type}(D_i)$. An attribute *value* a_i is then a specific valid value from the respective domain of the attribute A_i.

3) Tuple (Row): An $n-$*tuple* in a relation $R_{/n}$ is a tuple[3] of attribute values $t_i = (a_1, a_2, \ldots, a_n)$, where a_j represents the value of the attribute A_j in R. The relation can thus be defined extensionally by the *unordered* set of its tuples: $R = \{t_1, t_2, \ldots, t_m\}$, corresponding to the *rows* of the table T_R.

4) Integrity constraints: Besides the domain constraints $\mathsf{dom}(A_i)$, the most important integrity constraints are the primary and foreign *keys*. A *primary* key PK of a relation R is a minimal subset of its attributes $R[PK] \subseteq \mathcal{A}_\mathcal{R}$ that uniquely identifies each tuple:

$$\forall t_1, t_2 \in R: \ (t_1[PK] = t_2[PK]) \Rightarrow (t_1 = t_2)\,.$$

A *foreign* key FK_{R_2} in relation R_1 then refers to the primary key PK of another relation R_2 as

$$\forall t \in R_1: \ t[FK] \in \{t'[PK] \mid t' \in R_2\}\,.$$

This constitutes the inter-relations in the database, with the RDBMs managing the *referential integrity* of $T_{R_1}[FK] \subseteq T_{R_2}[PK]$.

B. Deep Learning

Deep learning [43] is a paradigm characterized by the use of *gradient descent* to optimize parameters of nested functions, commonly viewed through their computation graphs, referred to as *neural networks*. The main conceptual idea lies in learning *latent representations* of the data corresponding to the inner layers of the networks, generally constrained to the form of fixed-size numeric tensors, which restricts directly applying deep learning to relational databases. While passing beyond that limitation, we will generalize upon concepts known from two neural architectures that address two forms or related (simpler) structured representations of *sequences* and *graphs*.

1) Transformers: The Transformer [5] is a popular *sequence-to-sequence* model, relying primarily on the "attention" mechanism for inter-relating the given sequence tokens $x_1, \ldots, x_n$. Each input token x_i here is *embedded* into a continuous vector representation: $E(x_i) \in \mathbb{R}^d$, and combined with a "positional encoding" capturing its positional role: $E'(x_i) = E(x_i) + \mathsf{pos}(x_i)$. The *self-attention* mechanism then inter-relates all pairs of the input tokens to update their values as

$$X' = \mathsf{attn}(Q, K, V) = \mathsf{softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) V\,,$$

where Q, K, and V are the so-called "*query*", "*key*", and "*value*" matrix projections ("roles") of the input embeddings $E'(X)$. This efficient matrix computation can (optionally) be further repeated in parallel with separate Q, K, V projection matrices (multi-head attention).

In addition to the self-attention, Transformers employ *cross-attention* for tasks involving two distinct streams of sequences. In cross-attention, the query matrix Q is derived from the target (t) sequence decoder's input, while the key K and value V matrices are derived from the source (s) sequence encoder's output as $X' = \mathsf{softmax}\left(\frac{Q_t \cdot K_s^T}{\sqrt{d_k}}\right) V_s$. In either case, the updated values X' then position-wise pass through two standard feed-forward network (FNN) layers: $\mathsf{FNN}(x'_i) = W_2 \cdot \mathsf{ReLU}(W_1 \cdot x'_i + b_1) + b_2$, followed by layer normalization to reduce internal covariate shift, and residual connections for improved gradient propagation.

2) Graph Neural Networks: GNNs are a general class of neural models aimed at *graph-structured* data using the concept of (differentiable) *message-passing* [26]. Given an input graph $G = (\mathcal{V}, \mathcal{E})$, with a set of nodes $\mathcal{V}$ and edges $\mathcal{E}$, let $h_v^{(l)} \in \mathbb{R}^{d^{(l)}}$ be the vector representation (embedding) of node v at layer l. The general concept of GNNs can then be defined through the following sequence of functions:

(i) *Message* function $M^{(l)} : \mathbb{R}^{d^{(l)}} \times \mathbb{R}^{d^{(l)}} \to \mathbb{R}^{d_m^{(l)}}$ computes messages for each edge $(u, v) \in E$:

$$m_{u \to v}^{(l)} \ = \ M^{(l)}(h_u^{(l)}, h_v^{(l)})\,.$$

(ii) *Aggregation* function $A^{(l)} : \{\mathbb{R}^{d_m^{(l)}}\} \to \mathbb{R}^{d_m^{(l)}}$ aggregates the messages for each $v \in V$:

$$M_v^{(l)} \ = \ A^{(l)}\left(\{m_{u \to v}^{(l)} \mid (u, v) \in E\}\right)\,.$$

(iii) *Update* function $U^{(l)} : \mathbb{R}^{d^{(l)}} \times \mathbb{R}^{d_m^{(l)}} \to \mathbb{R}^{d^{(l+1)}}$ updates representation of each $v \in V$:

$$h_v^{(l+1)} \ = \ U^{(l)}(h_v^{(l)}, M_v^{(l)})\,.$$

The particular choice of the message, aggregation, and update functions then varies across specific GNN models, which are commonly composed of a predefined number L of such layers, enabling the message-passing to propagate information across L-neighborhoods within the graph(s). Note that the attention module of the Transformer follows the same schema while assuming a fully connected graph.

IV. Proposed Architecture

In this section, we describe the proposed learning representation and the relational message-passing architecture designed for end-to-end deep learning of Transformers from databases.

A. Data and Learning Representations

1) Nested hypergraphs: In order to directly follow the inductive bias of the relational database model (Sec. III-A), we consider the learning representation of a database as a *two-level multi-relational hypergraph*, where (i) each relation $R_{/n}$ forms n-ary hyperedges corresponding to the n-tuples *intra-relating* its attributes $\{t_i = (a_1, a_2, \ldots, a_n)\}$, and (ii) each

[3] the ordering is instantiated through the naming of the attributes

pair R_1, R_2 of such relations *inter-related* through the foreign key constraints $R_1[FK_{R_2}] \subseteq R_2[PK]$ forms another set of hyperedges from the respective tuple pairs $\{(t^1 \cup t^2) \mid t^1 \in R_1, t^2 \in R_2,\ t^1[FK_{R_2}] = t^2[PK]\}$. Note we consider all the tuple attributes $(a_1^1, \ldots, a_n^1 \cup a_1^2, \ldots, a_m^2)$ to form the link, and not just their keys, as these may also be composite, possibly spanning the whole tuple as a corner case of $R[PK] = \mathcal{A}_{\mathcal{R}}$, hence the forming of *hyperedges* instead of just edges here.

Additionally, for each such foreign-key tuple pair (t^1, t^2), we also consider the "reverse" hyperedge (t^2, t^1) to be able to fully propagate learning representations throughout the database, irrespective of the (ad-hoc) ordering choices of the database designer.

We then use the tuple pairs of (t^1, t^2) and (t^2, t^1) to build a bi-directional bi-partite hypergraph, connecting the tuples of the individual relations $R_{1/n}, R_{2/m}$, for each foreign key constraint in the database schema.

2) Schema detection: We aim at direct deep learning from raw database storage systems with as little preprocessing as possible while retaining the proper relational model semantics [42], for which we consider the relations' attribute values a_i as the minimal processing unit, building on the formal assumption of *atomicity* [42]. However, the current RDBMSs do not preserve the respective attribute type semantics required for deep learning. For instance, for integer-type ("int") columns, the information on whether the data contained are of nominal, ordinal, or cyclic nature is missing. Similarly, string-type ("varchar") columns may either contain actual text or encode discrete categories. However, such information is crucial to properly process the data with the neural models.

A distinction must also be made about attributes that form the key constraints as to whether they convey actual information or serve merely the referential purpose. To resolve such issues while avoiding manual data preprocessing, we have built an automated procedure that attempts to determine all such information from the database schema based on a combination of simple heuristics and selected data statistics. Once the schema (Sec. III-A) is detected with all the attribute $A_i \in \mathcal{A}$ types $\texttt{type}(D_i)$ determined, we first proceed with their *encoding* to numerical values. Notably, we (optionally) transform the textual types with a pre-trained language model, particularly Sentence-BERT [44] (App. C).

We then continue with *embedding* of the attributes in an appropriate fashion. Particularly, following methods from the tabular Transformers (Sec. II), we use a simple lookup table that stores embeddings of the detected categorical types, and "stack" or "linear" embedding of the numeric types (see App. C-A for details). Additionally, we (optionally) include the cyclic ("date/time") types with a special embedding respecting the periodic structure of the timestamp [45]. Importantly, each attribute has its own embedding function to allow for separate latent spaces.

3) Data loading: For machine learning, we need to establish what constitutes the learning samples (x_i, y_i) in the given relational setting. In this paper, we consider the standard (self-)supervised scenario where a single attribute A_j of a single target relation R forms the output labels y_i. Nevertheless, in contrast to the (classic) tabular setting, the input examples x_i can no longer be considered as i.i.d. tuples.

There are generally two cases: either (a) the database contains separate relational samples where each row t_i of the target table T_R belongs to a single learning instance x_i, or (b) the database cannot be split into such separate components, with x_i possibly spanning the whole hypergraph structure. To extract batches of the learning samples (x_i, y_i), irrespective of the structure, we follow a simple breadth-first-search (BFS) procedure, starting from each row t_i of the target table T_R and expanding over all the tables related through the foreign key constraints, in both the referenced and the referencing directions, while checking for loops.[4]

4) Data sampling: A salient feature of relational databases is that they can be very large, for which we optionally allow to run the loading natively *in-database* through recursive SQL (self-)joins with which minibatches of the hypergraph samples $\{(x_i, y_i)\}$ may be fetched into memory in a lazy fashion (with caching) from the, possibly remote, RDBMS. To make sure that the resulting hypergraph samples fit into memory, particularly in the (b) case, we (optionally) bound the BFS with a depth limit.[5]

Nevertheless, in the (most) cases where the whole database simply fits into memory, the whole hypergraph structure can be conveniently loaded and accessed with the more flexible neighborhood *sampling* techniques [46]. Particularly, we utilize the heterogeneous graph sampling routine introduced in [47], which proved most suitable for our relational setting.

B. Neural Architecture Space

To natively facilitate deep learning on the two-level hypergraph structure of the relational model (Sec. IV-A), we introduce a general two-level neural message-passing scheme composed of modular differentiable parameterized operations defined on the levels of (i) individual *attributes* (ii) and (sets of) related *tuples*. We further divide these operations w.r.t. their input-output characteristics into three categories:

(i) standard *Transformations*

$$X \overset{1:1}{\mapsto} Y\,,$$

(ii) n-ary *Combinations*

$$(X_1, X_2, \ldots, X_N) \overset{N:1}{\mapsto} Y\,,$$

(iii) permutation-invariant *Aggregations*

$$\{X_1, X_2, \ldots, X_M\} \overset{M:1}{\mapsto} Y\,,$$

where X, X_i and Y may refer to either the attributes a or the tuples t. Note that this can be seen as an extension of the "message-aggregate-update" paradigm of the GNNs (Sec.III). An instance of the proposed scheme is outlined in Fig. 1.

[4]Due to the possible interdependence between the samples, care must be taken to prevent information leakage about the labels, for which we mask out all target labels from the target column A_j of T_R when processing the samples. Overlooking this precaution led to some inappropriate accuracy reports in some of the related works.

[5]In inductive learning settings, this limit can be set to correspond to the perimeter of the relational receptive field of the subsequent neural message-passing, corresponding e.g. to the number of layers in GNN models (Sec. III-B), without loss of information.

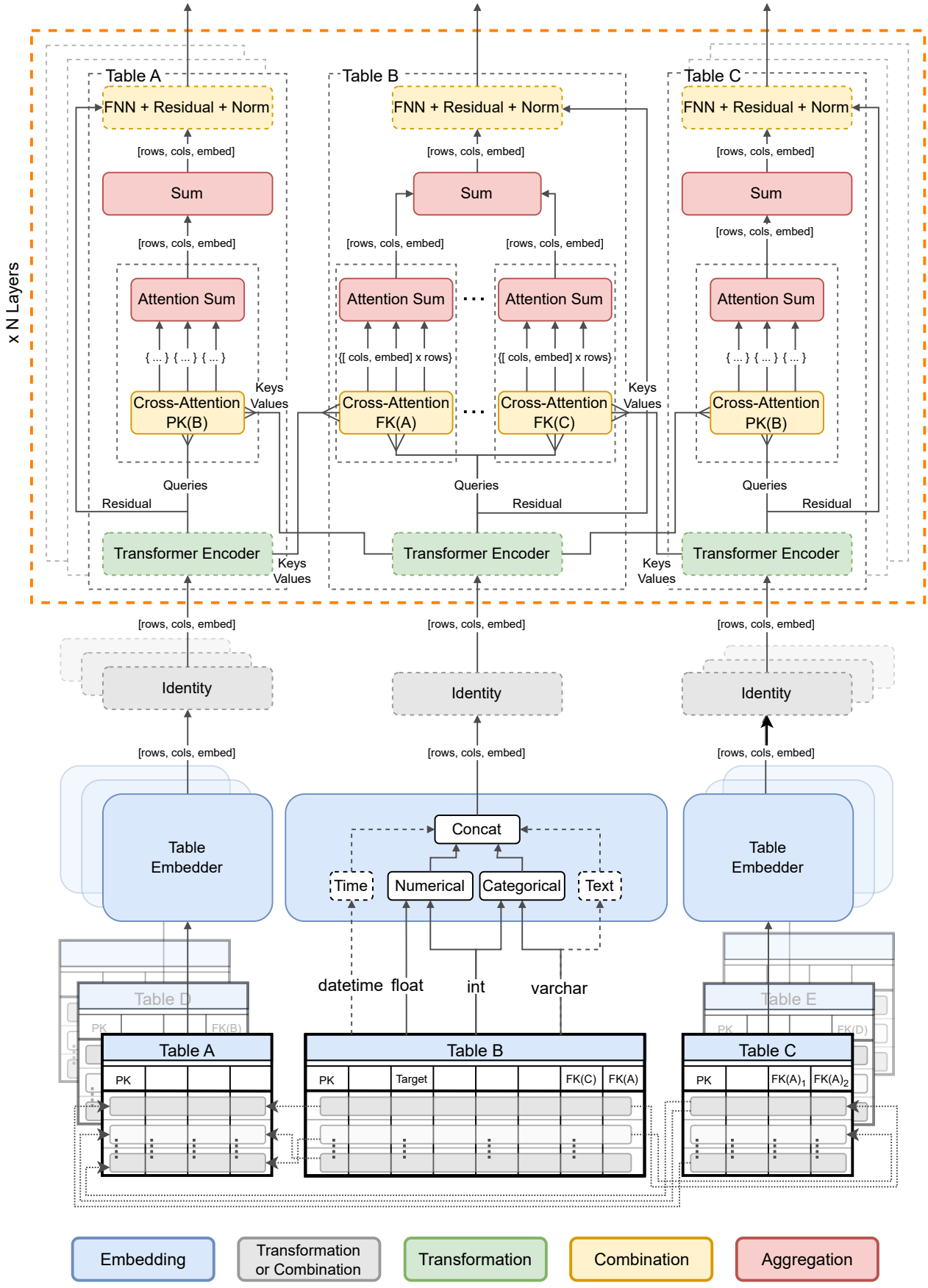

Fig. 1. The relational message-passing scheme of the proposed neural architecture space, instantiated with operations of the leading DBFormer model.

1) Architecture scheme: Every instantiation of the scheme starts with the embedding (Sec. IV-A) transformation ("Embedder") of the individual relation $R_{/n}$ attribute values $E(a_1), \ldots, E(a_n)$, resulting into an n-tuple of vectors $t_i^{(0)} \in (\mathbb{R}^{d_1}, \ldots, \mathbb{R}^{d_n})$ per each original tuple t_i from $R_{/n}$.[6] Each such tuple $t^{(0)}$ then undergoes *either* (i) an attribute *combination*

$$C_a : (a_1, a_2, \ldots, a_n) \overset{n:1}{\mapsto} (a') = t^{(1)}$$

that merges the attribute embeddings into a joint tuple embedding $t^{(1)} \in \mathbb{R}^{d_{t^{(1)}}}$ or (ii) a tuple *transformation*

$$T_t : (a_1, \ldots, a_n) \overset{1:1}{\mapsto} (a'_1, \ldots, a'_n) = t^{(1)}$$

that keeps the attribute embeddings separate as $t^{(1)} \in (\mathbb{R}^{d_{a^{(1)}}}, \ldots, \mathbb{R}^{d_{a^{(1)}}})$. In either case, the resulting tuple representation $t^{(1)}$ subsequently enters the second level of neural computation where it gets combined with all the tuples related through the second type of hyperedges (Sec. IV-A). Particularly, each $t_i^{(1)} \in R_1^{(1)}$ undergoes a tuple *combination*

$$C_t : (t_i^{(1)}, t_j^{(1)}) \overset{2:1}{\mapsto} t_{i_{R_2}}^{(2)} \in \mathbb{R}^{d_{t^{(2)}}}$$

with each $t_j^{(1)} \in R_2^{(1)}$, where $t_i[FK_{R_2}] = t_j[PK_{R_2}]$, resulting into a set of $\{t_{i_{R_2}}^{(2)}\}$ representations for each such pair of $t_i \in R_1$ and the related R_2. Each such set of the combined representations then undergoes a tuple *aggregation*

$$A_t : \{t_{i_{R_2}}^{(2)}\} \overset{m:1}{\mapsto} t_{i_{R_2}}^{(3)} \ ,$$

where $m = |\{t_{i_{R_2}}^{(2)}\}|$, to obtain one $t_{i_{R_2}}^{(3)}$ representation. Finally, we *aggregate* all such tuple representations

$$A_{t_R} : \{t_{i_{R_k}}^{(3)}\} \overset{l:1}{\mapsto} t_i^{(4)} \in \mathbb{R}^{d_{t^{(4)}}}$$

from all the $l = |\{R_k\}|$ linked relations back into a single final tuple representation $t_i^{(4)}$ for each $t_i \in R_{/n}$. Importantly, the same computation is performed simultaneously for *each* relation $R_{/n}$ in the database, and the resulting representations may be used again as input into subsequent layers of the same computation scheme in the classic spirit of deep learning.

2) Optional operations: Additionally, the scheme allows for optional intermediate blocks (dashed borders in Fig. 1).

First and foremost, this includes a "post-embedding" block that addresses the outlined division into the two options of (i) attribute combination C_a and (ii) transformation T_a in the first step of the scheme. Notably, combining the attributes in the (i) case disposes of the original column structure of $R_{/n}$, reducing the data dimensionality from $R^{n \times d}$ to R^d, and turning the remainder of the scheme into a largely standard single-level heterogeneous GNN computation [48], as explored in some of the related works (Sec. II). Such operation can range from a simple concatenation to *Tabular Transformers* that themselves combine columns into a single row embedding, such as Trompt [12] or TabNet [9]. Opting for the (ii) transformation then retains the original tabular structure throughout the scheme, for which we utilize operations ranging from simple positional encoding to tabular Transformer blocks retaining the columns, such as the SAINT [11] and TabTransformer [49].

The subsequent (optional) tuple *transformation* then follows the same logic while being *repeatedly* applied at the beginning of each layer of the scheme, for which the chosen model has to comply with the respective interface. Finally, the scheme allows for a closing (optional) tuple combination, facilitating a residual connection stream in the overarching relational part.

C. *The* DBFORMER

Technically, any differentiable parameterized operations that satisfy the corresponding input-output interface of the transformation, combination, and aggregation operators can be used in their respective places within the scheme, some of which are presented in our experiments (Sec. V). Nevertheless, we highlight one particular instantiation that we deem to most closely integrate the essence of the original Transformer architecture [5] with the relational database model (Sec. III), which we further refer to as the DBFORMER, depicted in Fig. 1.

Firstly, the model instantiates a Transformer Encoder in place of the tuple *transformation*, facilitating *self-attention* over the relations' attributes in the standard spirit of the tabular Transformers [8], but repeated across the database and over the layers, as part of the relational scheme. Secondly, the model also uses *cross-attention* in place of the tuple *combination* as

$$C_t(t_i, t_j) = \mathsf{attn}(Q = t_i, K = t_j, V = t_j) \, ,$$

essentially forming a Transformer Decoder from the remaining part of the scheme per each pair of interrelated relations.

We hypothesize that the cross-attention module used in this place might be able to extract the necessary *latent* relational features, as exploited with the successful propositionalization methods (Sec. II), but in a fully end-to-end fashion through gradient descent. Based on the notable expressiveness of Transformers [50], the select-join-aggregate operations normally used to construct such relational features should be well within the hypothesis space of the resulting architecture, in which we assume the query, key, and value roles of the input tokens to correspond to the foreign-key, primary-key, and column-value roles of the individual attributes, respectively. The idea is that the self-attention firstly transforms the tuple attributes w.r.t. each other within the tables, the cross-attention then learns their contextual interactions with attributes from the referenced tuples, and the attention-sum finally weights all their importance w.r.t. the referencing tuples.

V. EXPERIMENTS

We test[7] a number of instantiations of the proposed scheme against representative models from the distinct related work categories (Sec. II) through standard supervised classification and regression tasks across a wide range of diverse relational database datasets.

[6] The attribute embedding dimensions $d_1, \ldots, d_n$ within and across the relations may generally differ, so as to accommodate the possibly varying information loads in the tables, but in this paper we set them to be the same for simplicity.

[7] The source code for the experiments can be found at https://github.com/jakubpeleska/deep-db-learning and the web server serving the database datasets is made publicly available at https://relational.fel.cvut.cz/

TABLE I
CLASSIFICATION ACCURACIES OF THE TESTED SCHEME INSTANTIATIONS, COMPARED AGAINST THE REPRESENTATIVE MODELS FROM THE RELATED AREAS (SEC. II) OVER A RANGE OF RELATIONAL DATABASE BENCHMARKS [51].

Category	Tabular	Relational	Propos.	Ne-Sy	Deep Relational (ours)					
Dataset / model	**FNN**	**RDNboost**	**getML**	**CILP**	**DBFORMER**	**GNN**	**TabNet**	**Trompt**	**TabTrans.**	**SAINT**
Carcinogenesis	N/A	59.18	47.96	69.39	**75.51**	69.39	73.47	69.39	70.41	72.45
CraftBeer	11.38	0.60	5.39	11.38	**58.08**	14.97	14.97	13.17	13.77	13.17
Dallas	49.23	49.23	**86.15**	83.08	61.54	55.38	66.15	58.46	56.92	56.92
financial	75.49	N/A	**97.06**	79.90	88.73	78.39	79.41	78.92	75.98	74.06
Mondial	N/A	39.34	N/A	N/A	**100.00**	93.44	96.72	98.95	94.07	96.72
MuskSmall	N/A	40.74	74.07	81.48	96.30	96.30	96.30	**100.00**	88.89	88.89
mutagenesis	96.43	83.93	80.36	92.86	96.43	**98.21**	**98.21**	96.43	96.43	94.64
Pima	N/A	68.70	N/A	N/A	**83.04**	80.43	83.48	80.87	80.00	81.30
PremierLeague	59.87	34.21	61.40	73.68	**99.53**	82.49	71.69	59.76	66.25	59.18
Toxicology	N/A	56.86	63.73	67.65	**73.53**	70.59	**73.53**	71.57	71.57	71.57
UW_std	92.79	91.57	69.88	66.27	**97.37**	98.06	86.73	85.98	86.90	93.39
WebKP	N/A	N/A	**59.70**	57.41	56.40	56.16	53.55	54.30	52.13	60.12
DCG	N/A	50.15	85.84	73.45	98.82	62.24	94.10	**100.00**	69.91	64.60
Same_gen	N/A	14.51	**100.00**	**100.00**	**100.00**	89.74	89.57	88.46	90.57	93.38
voc	78.88	50.02	N/A	N/A	**85.16**	79.13	67.90	68.59	76.34	74.58
PubMed	N/A	N/A	**85.51**	84.87	63.38	55.22	52.48	61.62	64.07	61.56
Accidents	77.40	N/A	N/A	N/A	**93.20**	78.70	77.43	78.22	78.16	77.75
imdb_ijs	64.23	37.19	**94.39**	94.36	93.29	63.73	64.04	63.27	64.16	63.51
tpcd	20.90	N/A	N/A	N/A	**73.35**	22.60	21.19	21.40	21.00	21.08
Avg. Rank	**7.53**	**8.58**	**5.84**	**5.58**	**1.95**	**4.37**	**4.11**	**4.74**	**5.00**	**5.11**

A. Datasets

While RDBMs are some of the most widespread data storages, publicly available relational database benchmarks are considerably scarce. There are numerous collections of classic tabular [52] and structured datasets [53], [54], including graphs [55], [56], some of which are conceptually close to the database setting [57], [58]. Nevertheless, none of these provide *actual* relational database representations.[8] Thus, as part of this work, we have re-established the most complete resource collection in this area, originally created by [51], where we currently maintain over 50 of actual (SQL) database datasets from various domains, together with historical scoreboards and additional statistics.[9] In this paper, we narrow these down to 19 classification (Tab. III) and 16 regression (Tab. IV) datasets, filtering out (uninteresting) databases that are either too small or too trivial to fit. The remaining datasets are of highly diverse characteristics w.r.t. their sizes, schemas, structures, and application domains, as further detailed in App. B.

B. Related work models

As a baseline instance of the scheme, we consider a simple *tabular* FNN model [6] operating solely on the target table, i.e., ignoring all the inter-relations. This naive strategy is useful in revealing whether the given dataset task is indeed relational in nature or not. From the statistical *relational learning* (Sec. II), we choose the state-of-the-art RDN-boost [59], which, following the lifting strategy, can (very roughly) be seen as a relational generalization of the popular gradient-boosted trees [3]. As the *propositionalization* representative, we select the FastProp algorithm followed by XGBoost [60] – a battle-proof combination as promoted in [20], which leads a number of the relational dataset scoreboards [51]. To cover the *neuro-symbolic* area, we further emulate the popular CILP++ method [25] by connecting propositionalization with a FNN model in a similar fashion. We were unable to put any of the few recent deep relational learning proposals (Sec. III) into operation, but some of the closest GNN-based works can be viewed as conceptually close to the reduced (attribute combination) variants of the scheme (Sec. IV-B).

[8] Instead, they present simplified CSV, JSON, or XML files that do not fully represent the RDBM setting.

[9] This collection also covers most of the previous benchmarks from the domain of relational learning (Sec. II).

C. Scheme instantiations

As the space of all the possible neural models within the proposed scheme is very large, we tested only a few selected instantiations. This means selecting some particular parameterized differentiable operations in place of the initial Embedder module, and the attribute a and tuple t transformations $T_{a/t}$, combinations $C_{a/t}$, and aggregations A_t (Sec. IV-B).

1) DBFORMER*:* This model, already detailed in Sec. IV-C and Fig. 1, consists of N layers where each can be defined as $C_t^{FNN+Norm} \circ A_t^{Sum} \circ A_t^{Attn} \circ C_t^{Cross-Attn} \circ T_t^{Trans.-Encoder}$.

With this instantiation, we further tested extending the initial baseline Embedder (Sec. IV-A), transforming merely the *categorical* and *numerical* values to embedding vectors with the use of lookup tables and linear transformations respectively, with a number of ablations described in detail in App. C.

2) DB GNN: This model can be seen as a "reduced" version of the proposed scheme for its use of the attribute-combination function that flattens the columns' dimension as $C_a : (a_1, \ldots, a_n) \mapsto (a_1 \ldots a_n)$, where $(a_1, \ldots, a_n) \in (\mathbb{R}^D, \ldots, \mathbb{R}^D)$ and $(a_1 \ldots a_n) \in \mathbb{R}^{n*D}$. The reduced dimensionality then allows for the use of standard graph convolution modules. Particularly, we employed the SAGE [61] convolution, with which the N repeating layers can be described as

$$A_t^{Sum} \circ A_t^{Sum} \circ C_t^{SAGEConv} \circ T_t^{BatchNorm+ReLU}.$$

The model uses the baseline Embedder, and the residual combination module is skipped.

TABLE II
REGRESSION NRMSE OF THE TESTED SCHEME INSTANTIATIONS, COMPARED AGAINST THE REPRESENTATIVE MODELS FROM THE RELATED AREAS (SEC. II) OVER A RANGE OF RELATIONAL DATABASE BENCHMARKS [51].

Category	Tabular	Propos.	Ne-Sy	Deep Relational (ours)					
Dataset / model	**FNN**	**getML**	**CILP**	**DBFORMER**	**GNN**	**TabNet**	**Trompt**	**TabTrans.**	**SAINT**
Biodegradability	0.1873	0.2061	0.2490	**0.1544**	0.1773	0.1701	0.1654	0.1798	0.1584
classicmodels	0.5752	0.6461	1.1939	0.5023	0.4877	0.4606	**0.4048**	0.4646	1.0870
GOSales	N/A	N/A	N/A	0.4179	0.5329	**0.3996**	0.5194	0.7880	0.7457
northwind	1.1036	1.1588	1.3597	**0.4816**	0.7387	0.8007	0.8784	0.8620	0.9749
Triazine	N/A	0.1962	0.1781	0.1354	0.1648	**0.1174**	0.1687	0.1752	0.1357
Basketball_men	**0.2043**	0.2283	0.2546	0.2271	0.2275	0.2798	0.2076	0.2474	0.2569
restbase	0.1915	0.1920	0.1989	0.1771	0.1872	**0.1685**	0.1834	0.1847	0.1827
AdventureWorks2014	0.0323	0.0453	3.2931	**0.0113**	0.0635	2.1720	2.9907	0.3383	2.3792
FNHK	0.8262	**0.6482**	0.6899	0.7965	0.7974	0.7277	0.8010	1.0024	0.7494
sakila	0.5447	N/A	N/A	0.5178	0.4913	**0.4654**	0.5525	0.5565	0.5242
stats	0.9488	2.5927	6.4693	**0.1410**	1.6549	0.2856	2.9517	3.0027	2.9768
Grants	2.4317	N/A	N/A	3.7295	3.7527	**2.4288**	3.0689	2.6871	3.2923
ConsumerExpenditures	6.3763	6.2638	7.368	6.3568	6.3594	**6.3380**	6.6393	6.7533	6.7640
employee	0.2691	N/A	N/A	**0.2644**	0.2645	0.4984	0.2650	0.2646	0.7050
SalesDB	N/A	N/A	N/A	**0.4167**	0.5145	0.5463	0.5076	0.4424	0.5474
Seznam	5.3442	N/A	6.1318	3.6561	3.9379	4.6834	4.3157	**3.4137**	4.0425
Avg. Rank	**5.5000**	**6.3125**	**7.8125**	**2.4375**	**4.0625**	**3.3125**	**4.5000**	**5.0625**	**5.4375**

3) DB Trompt: This instance is designed to closely follow the tabular architecture of Trompt, as introduced in [12]. The Trompt Encoder is used once at the beginning as the "post-embedding" (Sec. IV-B) module to transform the data. The N repeating layers then have a simple definition of

$$A_t^{Sum} \circ A_t^{Sum} \circ C_t^{AddMean} ,$$

where the tuple transformation and closing combination modules are skipped, and

$$C_t^{AddMean}(t_i, t_j) = t_i + \frac{1}{dim(t_j)} \sum_{a_k \in t_j} a_k . \qquad (1)$$

Notably, the model utilizes the Trompt Decoder as a prediction head and has a custom Embedder that extends the baseline by following the categorical embeddings with Layer Normalization [62]. It also uses linear transformation of numerical values followed by a ReLU activation and Layer Normalization.

4) DB TabNet: Another tested instance based on a tabular Transformer is the DB extension of TabNet [9]. The TabNet encoder is formed by a series of repeated Feature Transformers, each followed by the Attention Transformer.[10]

Similarly to the DB GNN, TabNet belongs to the "reduced" category. Its Embedder processes only the *categorical* variables through the embeddings lookup table, and the *numerical* variables are duplicated to the target dimension by the Stack Embedder (App. C). Its N repeated layers can be defined as

$$A_t^{Sum} \circ A_t^{Sum} \circ C_t^{AddMean} \circ T_t^{TabNet-Encoder} ,$$

where $C_t^{AddMean}$ is defined in Equation 1.

5) DB SAINT: The SAINT instance refers to the tabular model introduced in [11]. The model takes a Transformer Encoder layer and extends it by a second block that uses "Intersample Attention," the details of which can be found in the article [11].

[10]For further description of the Feature and Attention Transformers, we refer to the original article [9].

The scheme's instance utilizes the "SAINT Encoder" layer as the tuple transformation operation in a mixture with the *cross-attention* for the tuple combination. The model also uses the baseline Embedder with an extension that a ReLU activation function follows the linear transformation. The N repeated layers can be defined as

$$C_t^{FF+Norm} \circ A_t^{Sum} \circ A_t^{Attn} \circ C_t^{Cross-Attn} \circ T_t^{SAINT-Encoder} .$$

6) DB TabTransformer: The last experimental instance is based on the TabTransformer [10] model. The TabTransformer architecture preprocesses only the *categorical* attributes, while *numerical* attributes are simply passed through Layer Normalization. The *categorical* columns are then passed through a Transformer Encoder block.

Similarly to the TabNet instance, the Embedder uses lookup table embeddings for *categorical* attributes and a Stack Embedder for *numerical* attributes to avoid transformations of the values. The rest of the N repeating layers are defined as follows

$$A_t^{Sum} \circ A_t^{Sum} \circ C_t^{AddMean} \circ T_t^{Trans.-Encoder/LayerNorm}$$

with $C_t^{AddMean}$ defined in Equation 1.

D. Parameterization

We follow a largely standard parameterization routine across all the methods. For the propositionalization-based related work, the number of relational features ranges around 200, depending on the depth of a custom BFS procedure that we implemented to improve their default performance, and the boosting works with the optimized default of $lr = 0.1$ and 100 base estimators. For the neural methods, including the baseline tabular FNN, we follow a standard deep learning setup of tuning the embedding dimensions, learning rate, and batch size, detailed further in App. A-B.

E. Results

Our classification and regression results with the models (Sec. V-B, V-C) are summarized in Table I and Table II,

respectively. Firstly, we see that many of the datasets are simply not accessible (N/A) to the tabular models (Tabular), in cases where the target table does not contain any informative attributes. Nevertheless, in the few cases where it does, even simple tabular models (FNN) perform very well, in accordance with [6]

The RDN-boost is a sophisticated SRL (Relational) method that does capture the relational inter-dependencies for which it, however, needs to set up "modes," [59] which we implemented in a rather straightforward fashion, possibly explaining its generally weaker performance. We note that we were unable to put the method into operation in the regression setting; hence, it is missing from the respective table. More importantly, the method does not scale well to larger datasets, reported (also) with the missing values. This issue was partially shared with the other relational methods, too. The getML (Fastprop+XGBoost) system [20], on the other hand, performed very well out-of-box, validating the strength of the propositionalization (Propos.) practice [63]. Similarly, the propositionalization-based neuro-symbolic (Ne-Sy) approach of CILP++ [25] performed very strongly, too.

Finally, instantiations of the proposed scheme generally displayed superior performances, with a small number of exceptions where the propositionalization shone. The overall best results were displayed by the proposed DBFORMER model (Sec. IV-C), demonstrating the strength of the close integration between the original Transformer architecture and the relational model. Nevertheless, the GNN instantiations, as well as the Tabular Transformer integrations with Trompt [11] and TabNet [9], exhibited strong performances, too.

VI. CONCLUSIONS

We introduced a general scheme that extends Transformers for deep learning from relational databases, utilizing a custom message-passing mechanism that adheres to the relational model of the common RDBMS. Our experiments with various instantiations of the scheme demonstrate its viability and superior performance as compared to commonly used methods from the associated fields of relational learning.

To improve the performance even further, incorporating self-supervised pre-training, in the spirit of the tabular models (Sec. II), for domain transfer across different *databases* seems like a promising avenue for future work.

APPENDIX A
EXPERIMENTAL SETUP

The output of the last layer, produced for the target table, is flattened if necessary and processed by a FNN prediction head with M layers, with each of the hidden layers followed by ReLU activation and, optionally, Batch Normalization [64]. For the standard gradient descent training in the classification tasks, the FNN output feeds into cross-entropy loss and MSE loss for the regression tasks, respectively.

For the metrics used in the results reporting, we simply leverage accuracy for the classification tasks and, to provide a somewhat comparable metric, a "Normalized Root Mean Squared Error" (NRMSE) is used across the regression tasks. The $NRMSE$ function is defined as

$$NRMSE(y, \hat{y}) = \frac{RMSE(y, \hat{y})}{\bar{y}}\,, \tag{2}$$

where $\bar{y}$ is the mean of all the training target values, and $RMSE$ function is defined as

$$RMSE(y, \hat{y}) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n}}\,. \tag{3}$$

A. Environment

All the executed experiments discussed in Section V used a simple hyperparameter optimization pipeline. The pipeline consisted of Ray [65], used for the distribution of resources and model training management; Optuna [66], used for searching over the hyperparameter space; and MLFlow [67], used for aggregating the parameters and metrics.

As for hardware, the training runs were split into two categories based on the dataset size, more precisely based on the number of rows in the target table (App. B). The runs on the datasets with less than or equal to 10,000 rows were trained on a single core of the `AMD EPYC 7742 64-Core` Processor and runs on larger datasets were executed on `NVIDIA A100-SXM4 40GB` GPU with a maximum of 4 runs sharing a single GPU.

B. Hyperparameters

There were 16 runs per model and dataset executed as part of the hyperparameter search, each running for 4000+ training steps[11] on a standard 70:30 training-validation split. All the neural models used vanilla Adam [68] optimizer with a learning rate set as a hyperparameter on a logarithmic space within $\langle 0.00005, 0.002 \rangle$. The heterogeneous graph sampling routine (HGSampling), as described in Section IV-B), facilitated the data sampling where the batch size was parametrized by the dataset size, with a hyperparameter scale factor from an exponential space in the interval $\langle 1, 2^8 \rangle$, and limited to a value of B, where $B \in 2^n$ and $n \in 4, 5, \ldots, 14$; hence the batch size always remained in the interval of $\langle 16, 16384 \rangle$. The embedding dimension D was also a hyperparameter in the search space, defined as a choice from the set of $\{16, 32, 64\}$. The number of layers N inside the scheme's instances was set as a random integer from $\{1, 2, 3, 4, 5\}$. The decision-making decoder FNN head was parametrized by the number of linear layers M that was 1, 2, or 3, where each hidden layer had 64 channels and a flag whether to use the "Batch Normalization."

APPENDIX B
DATASETS

The database datasets [51] used for the classification and regression tasks can be viewed in Tables III and IV, respectively. The tables contain statistics about the relational databases that they represent: 'Num. Rels.' - number of relations inside the

[11] With an exception of models that reached a hard training limit of 2 hours, however, this limit was surpassed on only the most extensive datasets such as "tpcd" (App. B) with large models. Nevertheless, extending this limit possibly allows for future improvements.

database, 'Num. Edge. Types' - number of primary, foreign key pairs, 'Num. Targ. Cols.' - number of non-key columns in the target table, 'Avg. Targ. Edges' - the average number of references from a single target table row to other tables, 'Total Num. Rows' - the overall number of rows in all tables of the database, such as 'Total Num. Edges' - the overall number of primary, foreign key pairs between all tables of the database, 'Text Col.' - whether the database contains non-key text attribute, and 'Time Col.' - whether the database contains *datetime* attribute.

TABLE III
A LIST OF *classification* DATASETS USED IN THE EXPERIMENTS WITH THE RESPECTIVE STATISTICS.

Dataset	Num. Rels.	Num. Edge Types	Num Targ. Cols.	Avg. Targ. Edges	Total Num. Rows	Total Num. Edges	Text Col.	Time Col.
Number of rows in target table: 1 - 1000								
Carcinoge.	6	13	1	83.21	28.0k	64.1k	False	False
CraftBeer	2	1	2	4.32	2968	2410	True	False
Dallas	3	2	13	2.71	812	593	True	True
financial	8	8	4	1	1.1M	1.1M	True	True
Mondial	34	63	1	1	21.4k	43.0k	True	True
MuskSmall	2	1	1	5.17	568	476	False	False
mutagen.	3	3	4	26.03	10.3k	15.3k	False	False
Pima	9	8	1	8	6912	6144	False	False
Prem.Leag.	4	5	3	29.29	11.3k	31.8k	True	True
Toxicology	4	5	1	53.26	49.8k	92.5k	False	False
UW_std	4	4	4	1.49	712	604	False	False
WebKP	3	3	1	94.16	81.9k	82.6k	False	False
Number of rows in target table: 1001 - 10 000								
DCG	2	1	1	6.31	8258	7128	False	False
Same_gen	4	6	1	2	1536	2978	False	False
voc	8	7	21	2.58	29.1k	21.0k	True	True
Number of rows in target table: 10 001 - 100 000								
PubMed	3	2	1	52.36	1.1M	1.0M	False	False
Number of rows in target table: 100 001 - 1 000 000								
Accidents	3	3	19	2.87	1.5M	2.4M	True	True
imdb_ijs	7	6	2	4.20	5.6M	8.2M	True	False
tpcd	8	10	5	11	8.7M	27.2M	True	True

TABLE IV
A LIST OF *regression* DATASETS USED IN THE EXPERIMENTS WITH THE RESPECTIVE STATISTICS.

Dataset	Num. Rels.	Num. Edge Types	Num Targ. Cols.	Avg. Targ. Edges	Total Num. Rows	Total Num. Edges	Text Col.	Time Col.
Number of rows in target table: 1 - 1000								
Biodegrad.	5	5	2	20.02	21.9k	33.1k	False	False
classicmod.	8	7	2	1	3864	6846	True	True
GOSales	5	4	1	39.5	151k	188k	True	True
northwind	11	10	9	5.6	3308	7113	True	True
Triazine	2	1	1	6	1302	1116	False	False
Number of rows in target table: 1001 - 10 000								
Basketball	9	9	59	23.18	44.8k	62.7k	True	True
restbase	3	3	2	1.99	19.3k	28.4k	True	False
Number of rows in target table: 10 001 - 100 000								
Adv.Works	70	90	14	11.26	760k	1.2M	True	True
FNHK	3	2	10	49.9	2.1M	2.1M	True	True
sakila	16	22	2	3	47.3k	122k	True	True
stats	8	12	11	17.44	1.0M	1.6M	True	True
Number of rows in target table: 100 001 - 1 000 000								
Grants	12	11	9	6.47	3.0M	5.1M	True	False
Number of rows in target table: 1 000 001 - 10 000 000								
Consu.Ex.	3	2	5	1	2.2M	2.2M	False	False
employee	6	6	2	1	3.9M	4.0M	True	True
SalesDB	4	3	1	3	6.7M	20.1M	True	False
Seznam	4	3	2	1	2.7M	2.6M	False	True

APPENDIX C
DBFORMER ABLATION STUDIES

In this appendix section, we report the ablations performed with the main DBFORMER model. The ablations are aimed to assess the sensitivity of the results w.r.t. (i) the selection of the initial embedding and (ii) the selection of the hyperparameters.

A. Embedders

The initial processing of data can often significantly influence the effectiveness of a model. Building on the work done in the field of tabular models (Sec. II), there is a variety of possible approaches. The categorical variables are almost always encoded with a simple embedding lookup table, with the exception of the models that do not use categorical variables at all, e.g., Excelformer [69]. Nevertheless, for the other variable types, several options may be considered.

1) *Stack Embedder*: the simplest option to increase the dimensionality of the *numeric* attributes is to copy the value D types in the embedding vector, where D is the target dimension of the embeddings.
2) *Linear Embedder*: a linear layer with *no activation* function, one input channel, and D output channels is another common way to create the embedding vectors out of *numeric* variables.
3) *Text Embeddings Transcoder*: as discussed in Section IV-B, plain text data from the database can be processed by a pre-trained language model. While it is unlikely that the language model embedding dimension will match the set-out dimension D, a linear layer with no activation can again be leveraged to address the dimensionality difference.
4) *Timestamp Embedder*: the most sophisticated embedding we considered is to account for the possible periodical information that might be encapsulated by the year, month, day, etc., of the timestamp attributes, for which the embedder first uses cyclic encoding with a combination of positional encoding to dimension d, where $d < D$, and only then puts the output through the linear layer to get embeddings of dimension D.

The classic tabular Transformer models usually only take the opportunity to combine simple embedding for the *categorical* variables with either the Stack or Linear Embedder for the *numerical* variables. However, usage of the text and timestamp attributes can potentially lead to performance gains. The DBFORMER, representing the leading model of this paper, was thus further tested with an additional list of such embedding options as follows:

1) *Baseline (base)*: the embedder uses only *categorical* and *numerical* variables with a simple embeddings lookup table and a Linear Embedder.
2) *With Text (text)*: extends the baseline embedder with *text embeddings* transformed by the Text Embeddings Transcoder.
3) *With Time (time)*: extends the baseline embedder with datetime attributes transformed by the Timestamp Embedder.

TABLE V
COMPARISON OF THE BASELINE DBFORMER TO ITS VERSION UTILIZING THE TEXTUAL EMBEDDINGS. MODELS ARE ONLY COMPARED ON THE DATASETS CONTAINING TEXTUAL NON-KEY ATTRIBUTES.

Classification
Model accuracy in %

Dataset	Baseline	With Text	Improvement
CraftBeer	12.57	58.08	**45.51**
Dallas	55.38	56.92	**1.54**
financial	74.02	78.43	**4.41**
Mondial	98.94	98.02	-0.92
PremierLeague	74.79	90.91	**16.12**
voc	79.46	80.20	**0.74**
Accidents	77.56	78.30	**0.74**
imdb_ijs	64.12	93.29	**29.17**
tpcd	21.26	73.35	**52.09**

Regression
Model NRMSE

Dataset	Baseline	With Text	Decrease
classicmodels	0.50	0.50	0.00
GOSales	0.42	0.26	**-0.16**
northwind	0.48	0.67	0.19
Basketball	0.23	0.20	**-0.03**
restbase	0.18	0.07	**-0.11**
AdventureWorks	0.01	1.61	1.60
FNHK	0.80	0.81	0.02
sakila	0.52	0.48	**-0.03**
stats	0.14	0.69	0.55
Grants	3.73	4.12	0.39
employee	0.26	0.26	0.00
SalesDB	0.42	0.13	**-0.28**

TABLE VI
COMPARISON OF THE BASELINE DBFORMER TO ITS VERSION UTILIZING THE TIMESTAMP EMBEDDINGS. MODELS ARE ONLY COMPARED ACROSS THE DATASETS CONTAINING TIME ATTRIBUTES.

Classification
Model accuracy in %

Dataset	Baseline	With Time	Improvement
Dallas	55.38	61.54	**6.16**
financial	74.02	88.73	**14.71**
Mondial	98.94	100.00	**1.06**
PremierLeague	74.79	99.53	**24.74**
voc	79.46	85.16	**5.70**
Accidents	77.56	79.08	**1.52**
tpcd	21.26	21.49	**0.23**

Regression
Model NRMSE

Dataset	Baseline	With Time	Decrease
classicmodels	0.50	0.16	**-0.34**
GOSales	0.42	0.17	**-0.24**
northwind	0.48	0.10	**-0.38**
Basketball	0.23	0.17	**-0.06**
AdventureWorks	0.01	0.05	0.04
FNHK	0.80	0.06	**-0.74**
sakila	0.52	0.36	**-0.16**
stats	0.14	0.16	0.02
employee	0.26	0.25	**-0.01**
Seznam	3.66	4.15	0.49

Table V compares the performance of the baseline DBFORMER setting to the one leveraging the textual embeddings. As can be seen, the textual embeddings significantly improve the model performance, confirming the usefulness of the information present in the often overlooked textual attributes.

The recently proposed work of [57] heavily emphasized the time dimension in the relational database setting. To experimentally evaluate its importance, Table VI shows the comparison of the DBFORMER model utilizing the time attributes with the Timestamp Embedder to its baseline version. As can be seen, the Timestamp Embedder strongly improves the performance on almost all relevant datasets, again validating the importance of the information present in the time attributes. Employing both text and time attributes thus showed significant improvements in performance.

TABLE VII
CLASSIFICATION ACCURACIES OF HYPERPARAMETER OPTIMIZED DBFORMER* COMPARED TO THE INSTANCES WITH FIXED HYPERPARAMETERS.

Dataset	DBFORMER*	LARGE	MEDIUM	SMALL
Carcinogenesis	75.51	69.39	68.37	**73.47**
CraftBeer	58.08	50.30	52.69	44.91
Dallas	61.54	56.92	52.31	55.38
financial	88.73	**95.44**	**86.76**	**89.22**
Mondial	100.00	**100.00**	93.44	93.44
MuskSmall	96.30	85.19	92.59	**96.30**
mutagenesis	96.43	**96.43**	**96.43**	**94.64**
Pima	83.04	**80.43**	**81.30**	**80.43**
PremierLeague	99.53	95.45	**99.07**	**99.53**
Toxicology	73.53	**70.59**	68.63	69.61
UW_std	97.37	**97.09**	88.80	94.17
WebKP	56.40	**54.84**	**55.25**	**55.33**
DCG	98.82	**98.82**	**100.00**	92.63
Same_gen	100.00	**100.00**	87.98	88.40
voc	85.16	**81.47**	84.87	83.76
PubMed	63.38	**62.70**	57.33	48.15
Accidents	93.20	69.19	77.66	79.38
imdb_ijs	93.29	**93.33**	**93.32**	**93.15**
tpcd	73.35	**70.60**	67.00	69.17

B. Hyperparameter sensitivity

All the previous experiments were carried out with the utilization of the reported hyperparameter optimization (App. A-B). To test the robustness of the main DBFORMER architecture, we also present results without the hyperparameter tuning over three versions of the model listed below.

1) LARGE: embedding dimension = 64, scheme N layers = 4, attention heads = 4, decoder hidden layers = 2, decoder hidden channels = 64
2) MEDIUM: embedding dimension = 32, scheme N layers = 3, attention heads = 4, decoder hidden layers = 2, decoder hidden channels = 64
3) SMALL: embedding dimension = 16, scheme N layers = 2, attention heads = 2, decoder hidden layers = 2, decoder hidden channels = 32

All three models were trained with a learning rate of 0.0001 using the vanilla Adam optimizer. The dropout rate inside the attention modules was set to 0.1, and all the decoder heads utilized the Batch Normalization. The initial Embedder module did extend the baseline with both the Text Embeddings Transcoder and the Timestamp Embedder in all cases, with all the remaining settings (App. A) being fixed.

The results in Table VII show that the DBFORMER model keeps displaying superior results, even without the hyperparameter tuning, and demonstrates the robustness of the architecture. Notably, the LARGE model is within 3% of the accuracy of the optimized DBFORMER* model (highlighted in bold) on the majority of the classification datasets. The MEDIUM and SMALL models then performed adequately well,

even outperforming the DBFORMER* in a few cases where the hyperparameter optimization apparently did not find the best settings (highlighted by underlining).

REFERENCES

[1] G. James, D. Witten, T. Hastie, R. Tibshirani *et al.*, *An introduction to statistical learning*. Springer, 2013, vol. 112.

[2] T. Halpin and T. Morgan, *Information modeling and relational databases*. Morgan Kaufmann, 2010.

[3] J. H. Friedman, "Greedy function approximation: a gradient boosting machine," *Annals of statistics*, pp. 1189–1232, 2001.

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

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Neural Information Processing Systems*, 2017.

[6] A. Kadra, M. Lindauer, F. Hutter, and J. Grabocka, "Well-tuned simple nets excel on tabular datasets," *Advances in neural information processing systems*, vol. 34, pp. 23 928–23 941, 2021.

[7] L. Getoor and B. Taskar, *Introduction to Statistical Relational Learning*. The MIT Press, 08 2007. [Online]. Available: https://doi.org/10.7551/mitpress/7432.001.0001

[8] G. Badaro, M. Saeed, and P. Papotti, "Transformers for tabular data representation: A survey of models and applications," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 227–249, 2023. [Online]. Available: https://aclanthology.org/2023.tacl-1.14

[9] S. O. Arik and T. Pfister, "Tabnet: Attentive interpretable tabular learning," 2020.

[10] X. Huang, A. Khetan, M. Cvitkovic, and Z. S. Karnin, "Tabtransformer: Tabular data modeling using contextual embeddings," *CoRR*, vol. abs/2012.06678, 2020. [Online]. Available: https://arxiv.org/abs/2012.06678

[11] G. Somepalli, M. Goldblum, A. Schwarzschild, C. B. Bruss, and T. Goldstein, "Saint: Improved neural networks for tabular data via row attention and contrastive pre-training," 2021.

[12] K.-Y. Chen, P.-H. Chiang, H.-R. Chou, T.-W. Chen, and T.-H. Chang, "Trompt: Towards a better deep neural network for tabular data," 2023.

[13] A. Cropper, S. Dumančić, and S. H. Muggleton, "Turning 30: New ideas in inductive logic programming," *arXiv preprint arXiv:2002.11002*, 2020.

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

[15] J. H. Gallier, *Logic for computer science: foundations of automatic theorem proving*. Courier Dover Publications, 2015.

[16] S. Muggleton and L. De Raedt, "Inductive logic programming: Theory and methods," *The Journal of Logic Programming*, vol. 19, 1994.

[17] A. Kimmig, L. Mihalkova, and L. Getoor, "Lifted graphical models: a survey," *Machine Learning*, vol. 99, no. 1, pp. 1–45, 2015.

[18] S. Kramer, N. Lavrač, and P. Flach, "Propositionalization approaches to relational data mining," *Relational data mining*, pp. 262–291, 2001.

[19] The Alteryx, "Featuretools." [Online]. Available: https://www.featuretools.com

[20] The SQLNet Company GmbH, "getml." [Online]. Available: https://getml.com

[21] B. Hammer and P. Hitzler, *Perspectives of neural-symbolic integration*. Springer, 2007, vol. 77.

[22] T. Rocktäschel and S. Riedel, "Learning knowledge base inference with neural theorem provers," *Proceedings of the 5th Workshop on Automated Knowledge Base Construction (AKBC)*, pp. 45–50, 2016.

[23] A. Serafini, Luciano, Garcez, L. Serafini, and A. S. d'Avila Garcez, "Learning and reasoning with logic tensor networks," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10037, pp. 334–348, 2016.

[24] G. Šourek, V. Aschenbrenner, F. Železný, S. Schockaert, and O. Kuželka, "Lifted relational neural networks: Efficient learning of latent relational structures," *Journal of Artificial Intelligence Research*, vol. 62, pp. 69–100, 2018.

[25] M. V. Franca, G. Zaverucha, and A. Garcez, "Fast relational learning using bottom clause propositionalization with artificial neural networks," *Machine learning*, vol. 94, no. 1, pp. 81–104, 2014.

[26] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[27] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2008.

[28] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3558–3565.

[29] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.

[30] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[31] M. Cvitkovic, "Supervised learning on relational databases with graph neural networks," *arXiv preprint arXiv:2002.02046*, 2020.

[32] S. Liu, D. Vazquez, J. Tang, and P.-A. Noel, "Flaky performances when pre-training on relational databases with a plan for future characterization efforts," in *First Workshop on Pre-training: Perspectives, Pitfalls, and Paths Forward at ICML 2022*, 2022.

[33] J. Bai, J. Wang, Z. Li, D. Ding, J. Zhang, and J. Gao, "Atj-net: Auto-table-join network for automatic learning on relational databases," in *Proceedings of the Web Conference 2021*, 2021, pp. 1540–1551.

[34] X. Deng, H. Sun, A. Lees, Y. Wu, and C. Yu, "Turl: Table understanding through representation learning," *ACM SIGMOD Record*, vol. 51, no. 1, pp. 33–40, 2022.

[35] G. Gaur, R. Singh, S. Arora, V. Gupta, and S. Bedathur, "Teaching old db neu(ral) tricks: Learning embeddings on multi-tabular databases," in *Proceedings of the 6th Joint International Conference on Data Science & Management of Data (10th ACM IKDD CODS and 28th COMAD)*, ser. CODS-COMAD '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 87–94. [Online]. Available: https://doi.org/10.1145/3570991.3571041

[36] L. Vogel, B. Hilprecht, and C. Binnig, "Towards foundation models for relational databases [vision paper]," *arXiv preprint arXiv:2305.15321*, 2023.

[37] B. Hilprecht, K. Kersting, and C. Binnig, "Spare: A single-pass neural model for relational databases," *arXiv preprint arXiv:2310.13581*, 2023.

[38] M. Fey, W. Hu, K. Huang, J. E. Lenssen, R. Ranjan, J. Robinson, R. Ying, J. You, and J. Leskovec, "Position: Relational deep learning - graph representation learning on relational databases," in *Forty-first International Conference on Machine Learning*, 2024. [Online]. Available: https://openreview.net/forum?id=BIMSHniyCP

[39] J. You and G. Liu, "Beyond graphs: Learning with relational DBs," 2024. [Online]. Available: https://openreview.net/forum?id=ZQIgnzxMKJ

[40] H. Zhang, Q. Gan, D. Wipf, and W. Zhang, "Gfs: Graph-based feature synthesis for prediction over relational databases," *arXiv preprint arXiv:2312.02037*, 2023.

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

[42] E. F. Codd, *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[43] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[44] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

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

[46] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in neural information processing systems*, 2017, pp. 1024–1034.

[47] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," 2020.

[48] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 793–803. [Online]. Available: https://doi.org/10.1145/3292500.3330961

[49] X. Hu, W. Tang, C.-K. Hsieh, and S. Shi, "Tabtransformer: Tabular data modeling using contextual embeddings," *arXiv preprint arXiv:2012.06678*, 2020.

[50] D. Lindner, J. Kramár, M. Rahtz, T. McGrath, and V. Mikulik, "Tracr: Compiled transformers as a laboratory for interpretability," *arXiv preprint arXiv:2301.05062*, 2023.

[51] J. Motl and O. Schulte, "The ctu prague relational learning repository," *arXiv preprint arXiv:1511.03086*, 2015.
[52] D. Aha, "UCI Machine Learning Repository," UCI Machine Learning Repository, 1987.
[53] J. Berant, D. Deutch, A. Globerson, T. Milo, and T. Wolfson, "Explaining queries over web tables to non-experts," 2018.
[54] A. Johnson, L. Bulgarelli, T. Pollard, S. Horng, L. Celi, and R. Mark, "Mimic-iv (version 1.0)," 2020.
[55] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.
[56] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec, "Open graph benchmark: Datasets for machine learning on graphs," 2021.
[57] J. Robinson, R. Ranjan, W. Hu, K. Huang, J. Han, A. Dobles, M. Fey, J. E. Lenssen, Y. Yuan, Z. Zhang, X. He, and J. 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. [Online]. Available: https://openreview.net/forum?id=WEFxOm3Aez
[58] L. Vogel, J.-M. Bodensohn, and C. Binnig, "Wikidbs: A large-scale corpus of relational databases from wikidata," in *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. [Online]. Available: https://openreview.net/pdf?id=abXaOcvujs
[59] S. Natarajan, T. Khot, K. Kersting, B. Gutmann, and J. Shavlik, "Gradient-based boosting for statistical relational learning: The relational dependency network case," *Machine Learning*, vol. 86, pp. 25–56, 2012.
[60] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.
[61] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.
[62] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016.
[63] M.-A. Krogel, S. Rawles, F. Železný, P. A. Flach, N. Lavrač, and S. Wrobel, *Comparative evaluation of approaches to propositionalization*. Springer, 2003.
[64] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.
[65] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica, "Ray: A distributed framework for emerging ai applications," 2018.
[66] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.
[67] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe *et al.*, "Accelerating the machine learning lifecycle with mlflow." *IEEE Data Eng. Bull.*, vol. 41, no. 4, pp. 39–45, 2018.
[68] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[69] J. Chen, J. Yan, D. Z. Chen, and J. Wu, "Excelformer: A neural network surpassing gbdts on tabular data," 2023.