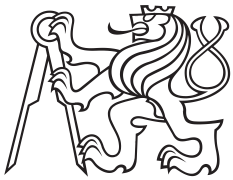
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Deep Learning for Relational Databases

Bc. Jakub Peleška

Supervisor: Ing. Gustav Šír, Ph.D.
Field of study: Open Informatics
Subfield: Data Science
May 2024

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Peleška Jakub** Personal ID number: **483727**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Data Science**

II. Master’s thesis details

Master’s thesis title in English:

Deep Learning for Relational Databases

Master’s thesis title in Czech:

Hluboké učení pro relační databáze

Guidelines:

Most machine learning algorithms target data in the form of numeric feature vectors or tensors, however, most of the real-world data are stored in relational databases. These could be addressed with specialized Relational Learning [5] algorithms which, however, do not scale well and lack most of the advantages of modern deep learning methods. Recently, transformer-based models, such as TabNet [1], have gained a lot of attention for learning from tabular data [2], i.e. in the format of a single table, where they are progressively narrowing the performance gap with established statistical methods like XGBoost [6]. The target of this project is a principled relational generalization of these models into setting with a set of interconnected tables, i.e. a relational database.
1) Study existing deep learning methods for tabular data [2].
2) Review prior art in relational learning [5] and its links to deep learning, such as Graph Neural Networks [4].
3) Explore possibilities for generalizing their principles to the relational database setting [3].
4) Propose integration of the resulting architecture with the existing deep tabular models.
5) Collect a significant amount of appropriate benchmark datasets.
6) Design a solid experimental workflow to test your proposed models across a variety of settings.

Bibliography / sources:

[1] Arik, S. Ö., & Pfister, T. (2021, May). Tabnet: Attentive interpretable tabular learning. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 35, No. 8, pp. 6679-6687).
[2] Borisov, Vadim, et al. "Deep neural networks and tabular data: A survey." IEEE Transactions on Neural Networks and Learning Systems (2022).
[3] Cvitkovic, Milan. "Supervised learning on relational databases with graph neural networks." arXiv preprint arXiv:2002.02046 (2020).
[4] Šourek, G., Železný, F., & Kuželka, O. (2021). Beyond graph neural networks with lifted relational neural networks. Machine Learning, 110(7), 1695-1738.
[5] Getoor, Lise, and Ben Taskar, eds. Introduction to statistical relational learning. MIT press, 2007.
[6] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016.

Name and workplace of master's thesis supervisor:

Ing. Gustav Šír, Ph.D. Intelligent Data Analysis FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **01.02.2024** Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Gustav Šír, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

Firstly, I would like to thank my supervisor, Ing. Gustav Šír, Ph.D., for the guidance and insightful suggestions during the work on the thesis.

Next, I would like to thank Ing. Jan Motl, Ph.D., for creating the CTU Prague Relational Learning Repository, whose help is greatly appreciated.

Last but not least, I sincerely thank everybody that supports me.

The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

Declaration

I declare that I have prepared the submitted thesis independently and that I have provided all information sources used in accordance with the Methodical instructions about ethical principles for writing academic thesis and the Framework Rules for the Use of Artificial Intelligence at CTU for Study and Pedagogical Purposes in BSc and MSc Studies.

.....

In Prague 24. 5. 2024

Jakub Peleška

Prohlašuji, že jsem předloženou práci vypracoval samostatně a uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací a Rámcovými pravidly používání umělé inteligence na ČVUT pro studijní a pedagogické účely v Bc a NM studiu.

.....

V Praze dne 24. 5. 2024

Jakub Peleška

Abstract

Relational Databases store the majority of the world's data. However, their use in deep learning is greatly underutilized. This thesis explores the integration of deep learning with relational databases, leveraging the intricate interconnections of the stored values.

Recent advancements in AI, particularly in deep learning models like transformers and CNNs, have revolutionized fields of natural language processing and computer vision through their ability to process homogeneous data. Nevertheless, relational database data are inherently heterogeneous and structured, posing challenges for traditional deep-learning approaches.

This research addresses the obstacle of data representation by viewing relational databases as heterogeneous tabular graphs, aligning with recent successes in graph neural networks. The proposed blueprint lays down a foundation for deep learning on relational databases. The neural architecture space of the blueprint allows for the employment of existing tabular models and, importantly, the sequence processing transformers. The presented Database Transformer highlights the strength of this framework by displaying promising results that outperform existing state-of-the-art methods.

Keywords: deep learning, relational databases, transformers, relational models, graph neural networks, heterogeneous graphs, tabular data

Supervisor: Ing. Gustav Šír, Ph.D.

Abstrakt

Relační databáze uchovávají většinu světových dat. Nicméně jejich potenciál v hlubokém učení je značně nevyužitý. Tato diplomová práce zkoumá integraci hlubokého učení s relačními databázemi, využívajíc komplikovaná vzájemná propojení uložených hodnot.

Nedávné pokroky v AI, zejména v modelech hlubokého učení jako jsou transformery a CNN, revolucionalizovaly oblasti zpracování přirozeného jazyka a počítačového vidění díky své schopnosti zpracovávat homogenní data. Avšak data v relačních databázích jsou svou povahou heterogenní a strukturovaná, což představuje výzvy pro tradiční přístupy hlubokého učení.

Výzkum se zabývá překážkou v reprezentaci dat tím, že pohlíží na relační databáze jako na heterogenní tabulární grafy, čímž navazuje na nedávné úspěchy v oblasti grafových neuronových sítí. Navrhovaný blueprint poskytuje základy pro hluboké učení nad relačními databázemi. Obecnost navrhovaného blueprintu umožňuje využití stávajících tabulárních modelů a, co je důležité, také integaci transformerů silných na zpracování řetězců dat. Database Transformer vykazuje výborné výsledky, které překonávají současné špičkové metody a tím podtrhuje sílu tohoto frameworku.

Klíčová slova: hluboké učení, relační databáze, transformer, relační modely, grafové neuronové sítě, heterogenní grafy, tabulární data

Překlad názvu: Hluboké učení pro relační databáze

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Goals of the thesis 3

2 Foundation **5**
2.1 Relational Databases 5
2.1.1 Relational Database Management Systems 6
2.1.2 SQL 6
2.2 Deep Learning 7
2.2.1 Training methods 8
2.2.2 Variable Types 9
2.2.3 Attention Mechanism 10
2.2.4 Message-Passing 12

3 Related fields of study **15**
3.1 Learning from Tabular Data 15
3.1.1 Data Encoding Methods 17
3.1.2 Transfomer-based Models 17
3.2 Relational Learning 19
3.2.1 Statistical Relational Learning . . . 19
3.2.2 Propositionalization 19
3.2.3 Neuro-Symbolic Integration 20
3.2.4 Graph Neural Networks 20
3.2.5 Beyond Graph Neural Networks . 22

4 Deep Learning for Relational Databases **23**
4.1 Data Representation 23
4.1.1 Heterogeneous Tabular Graph . . 24
4.1.2 Schema Detection 25
4.1.3 Data Loading 26
4.1.4 Graph Construction 27
4.1.5 Data Sampling 27
4.2 The Blueprint 29
4.2.1 Modules 30
4.3 Blueprint Classes 33
4.3.1 Embedders 33
4.3.2 Inner Dimensionality 34
4.3.3 Tabular Models 35
4.4 Database Transformer 35
4.5 Experimental Blueprint Instances . . . 37
4.5.1 Database Transformer Versions . 37
4.5.2 SAGE GNN 38
4.5.3 SAINT 38
4.5.4 Trompt 39
4.5.5 TabNet 39
4.5.6 TabTransformer 40

5 Experiments **41**
5.1 Benchmark Datasets 41
5.1.1 Existing Benchmark Datasets 42
5.1.2 The CTU Prague Relational Learning Repository 42
5.2 Hyperparameter Optimization 45
5.2.1 Distributed Computing 45
5.2.2 Tuning 46
5.2.3 Aggregation 46
5.3 Experiment Runs 46
5.3.1 Tasks 47
5.3.2 Environment 49
5.3.3 Parametrization 50
5.3.4 Overall Results 50
5.3.5 Database Transformer Versions . 54

6 Conclusion **59**
6.1 Future Work 60
6.1.1 Blueprint instances 61
6.1.2 Enriching the RelBench Package . 61
6.1.3 Pre-training 61
6.1.4 Temporal Data 62
6.1.5 Attention encoding of SQL statements 62
6.1.6 Real-world application 63

Bibliography **65**

A Glossary **73**

B Supplementary material **75**

Chapter 1

Introduction

1.1 Motivation

In recent years, artificial intelligence (AI) has achieved significant milestones across various domains, demonstrating remarkable progress in both theoretical advancements and practical applications. Notable successes include the development of deep learning models, such as convolutional neural networks (CNNs) and transformers, which have revolutionized computer vision and natural language processing, respectively. AI-driven diagnostic tools have enhanced medical imaging, enabling exceptional accuracy in the early detection of diseases like cancer. Autonomous systems, such as self-driving cars, have improved safety and reliability, although challenges remain.

Amidst this evolving AI landscape, deep learning has emerged as a transformative and innovative force pushing the limits of AI capabilities. Its notable successes have primarily been driven by its immense ability to handle homogeneous data — data in the form of numeric feature vectors or tensors, such as images or text. This homogeneity allows for the application of uniform processing and analysis techniques, leading to significant advancements in the aforementioned fields, such as computer vision and natural language processing.

Although homogeneous data have proven to be an effective form of input for deep learning models, importantly due to the use of proven robust methods of linear algebra and utilization of specialized hardware, the real-world data

usually does not follow such form. This discontinuity between the AI model data representation and the original form of data is frequently handled by pre-processing routines. These pipelines, among others, use transformations, filtering, and mapping techniques to match the demanded shape of the model's input tensor. As a result, part of the information hidden within the data's original structure is lost.

Relational databases have been, for decades, a golden standard for data storage. Even today, with the rising NoSQL databases, the relational model stays strong as an established solution for handling valuable knowledge collected by universities, governments, healthcare institutions, and even leading tech companies. Most of the world's data is stored within these relational databases managed by numerous RDBMSs systems. As such, the majority of the world's data is *not homogeneous* nor in the form of tensors. This suggests a huge unleveraged potential as the relational *heterogeneous* graph-like structure is mostly unutilized.

Traditionally, feature engineering has been a critical step in transitioning from a relational database to a simpler, yet *heterogeneous*, tabular data format. It involves manually selecting and transforming data from these databases into a more straightforward form of a single table suitable for tabular machine learning models — a task that is often time-consuming, requires domain expertise, is prone to human bias, and fundamentally removes some of the structural information from the original database.

Yet, a disproportional amount of methods work with a simplified representation, not capturing the full potential of the relational model. However, this disproportionality can be accounted for by one major factor: the complexity of the *heterogeneous tabular graph* data format that represents values stored within the relational databases. This type of data comprises various data types and relations, making it more complex and less amenable to traditional deep-learning approaches. Recent breakthroughs in this area have started addressing these complexities, opening new avenues for effectively leveraging the rich and varied information in *heterogeneous tabular graph* data.

This approach aligns with the recent successes in the field of Graph Neural Networks (GNNs). GNNs have shown remarkable ability in handling data represented in graph structures, where nodes and edges can represent various interconnected entities and relationships. By viewing relational databases through this lens, it becomes possible to directly apply advanced deep learning techniques, bypassing the labor-intensive process of feature engineering. This

paradigm shift enables leveraging the inherent relational structure within relational databases, aligning with the strengths of GNNs in processing relational data.

1.2 Goals of the thesis

The primary objectives of this thesis are to advance the understanding and capabilities of deep learning technologies on relational databases viewed as *heterogeneous tabular graph*. Initially, the thesis will conduct a comprehensive review of these related fields, examining current methodologies, recent advancements, and their applications in various domains. A key component of this research involves the provision of public dataset benchmarks, which will facilitate direct comparisons and enhancements in model training processes.

Furthermore, the thesis aims to develop a general model structure designed for efficient learning from these RDBs. This *blueprint* will integrate the principles of tabular models, graph neural networks, and transformer architecture, enhancing their applicability and performance in handling complex structured relational data.

Finally, the effectiveness of the developed models will be evaluated by comparing their performance on provided dataset benchmarks against other state-of-the-art solutions in the field. This testing will not only validate the models' effectiveness but also highlight areas for potential improvement, setting a foundation for future research and development in deep learning on relational databases.

Chapter 2

Foundation

Before discussing new ideas and solutions, it is necessary to understand the premise of the thesis. To get started, let's review the two topics mentioned directly in the title - Relational Databases and Deep Learning.

2.1 Relational Databases

"A relational database is a type of database that organizes data into rows and columns, which collectively form a table where the data points are related to each other."[1] At its core lays the *relational model* that was first proposed by Edgar F. Codd in *"A Relational Model of Data for Large Shared Data Banks"*[1] that provides a unified way of data management independent of the software specifics of a given system. The key part of this feature is to view the data in the form of n-ary *relations* in the accepted mathematical sense. For the purposes of further topics in this work, let's define a couple of concepts as described by Edgar F. Codd[1, 2]. Terms in the parentheses may be used interchangeably with the associated main term.

- **Relation (Table)** - Given sets S_1, S_2, ..., S_n of values of n *attributes*, an n-ary relation $R_{/n}$ is a subset of the Cartesian product $S_1 \times S_2 \times$ ... $\times S_{n-1} \times S_n$. Relation R is usually presented as *table* T_R with column headings followed by rows of column values in a fixed order.

[1] https://www.ibm.com/topics/relational-databases

- **Attribute (Column)** - Attributes $\mathcal{A}_R = \{A_1, A_2, ..., A_n\}$ define the terms of a relation R, corresponding to the columns of the respective table T_R. Each attribute is a pair of names and types, constraining each attribute's domain $type(D_i) \subseteq dom(A_i)$. An attribute value a_i is a specific valid value from the respective attribute type A_i.
- **Tuple (Row)** - An $n-$tuple t_i in the relation $R_{/n}$ of attribute values $t_i = (a_1, a_2, ..., a_n)$, where a_j represents the value of the attribute A_j in $R_{/n}$.
- **Integrity Constrains** - The relational databases allow for connections between the relations via *primary* and *foreign* key pairs. A *primary* key PK of relation R_1 is a subset of its attributes $\mathcal{A}_{R_1}$ that uniquely identifies each tuple in the relation R_1; hence if $t_i[PK] = t_j[PK]$ then $t_i = t_j$. A *foreign* key FK_{R_2} of relation R_1 referencing a relation R_2 is a subset of attributes $\mathcal{A}_{R_2}$ representing a *primary* key of relation R_2; hence can be written as $\forall t_i \in R_1 \exists t_j \in R_2$ such $t_i[FK_{R_2}] = t_j[PK]$.

2.1.1 Relational Database Management Systems

Relational databases are also often associated with other topics, such as transactions (A.C.I.D. properties)[3], database normalization, and SQL, but these topics are rather connected to relational database management systems (RDBMSs). RDBMS is specific software that provides users access control, data retrieval, value validity, primary foreign key reference management, etc. Nevertheless, it is essential to mention them. They are an important factor that allowed the emergence and popularity of relational databases and all modern RDBMSs.

2.1.2 SQL

Structured Query Language (SQL) is a standardized programming language specifically designed for managing RDBMS and manipulating the underlying relational databases. Developed in the early 1970s by IBM researchers Donald D. Chamberlin and Raymond F. Boyce, SQL played a crucial part in the popularization of the RDBMSs. The functionalities provided by SQL can be categorized into several types of operations - Data Query Language (DQL), Data Definition Language (DDL), Data Manipulation Language (DML), and

Data Control Language (DCL). For the purposes of this work, it is not necessary to go into detail about the separate subparts of the SQL language. However, it should be mentioned that SQL provides users with various data types. A list of the most common ones is below.

- **Boolean Types** - `BOOLEAN`.
- **Numeric Types** - `INTEGER`, `FLOAT`, `DECIMAL`, etc.
- **Character Types** - `CHAR`, `VARCHAR`, `TEXT`, etc.
- **Date and Time Types** - `DATE`, `TIME`, `TIMESTAMP`, etc.

2.2 Deep Learning

Deep learning (DL)[4] has experienced a steep rise over the last decade[5], becoming one of the most dynamic areas within artificial intelligence. DL's origins trace back to the advent of artificial neural networks (NNs), and it can be speculated that first NNs were essentially variants of linear regression methods, putting the deep origins to the early 1800s[6]. Since the first DNNs[7], a truly gargantuan amount of work was put into the field [6] but only reasonably recent innovations in hardware[8] allowed for the advancement of learning algorithms and an explosion in data availability that have propelled deep learning from academic labs to the forefront of real-world applications, radically transforming industries from healthcare to automotive[9][10][11].

Deep neural networks, the fundamental building blocks of deep learning, are structured in layers, each composed of numerous interconnected units. Non-linear mathematical functions known as activation functions (e.g., ReLU[12], GeLU[13]) are often used between the layers to determine the output at each layer based on its input. Data flows through the network's layers in a forward pass, while errors are corrected during the backward pass through a back-propagation. This involves calculating the gradient of the loss function with respect to the network's weights and, using techniques like stochastic gradient descent [14] or, more recently, Adam [15], updating the weights to minimize the error.

Deep learning encompasses a variety of model types, each suited to different tasks. Convolutional Neural Networks (CNNs) [16] are pivotal in image

recognition and processing, whereas Recurrent Neural Networks (RNNs)[17] excel in handling sequential data like speech. More recently, attention-based models like Transformers[18] have become prevalent in processing sequences, particularly in natural language processing. This led to the development of large language models (LLMs). These models, including GPT (Generative Pre-trained Transformer)[19] and BERT (Bidirectional Encoder Representations from Transformers)[20], have revolutionized how machines understand and generate human language. Capable of performing tasks ranging from translation to content generation, LLMs demonstrate capabilities that continue to expand the boundaries of artificial intelligence. Additionally, Graph Neural Networks (GNNs)[21] have emerged to address data that is structured specifically as graphs. These networks leverage the relationships and interconnections between nodes in a graph, making them ideal for applications such as social network analysis, molecule structure analysis, and recommendation systems[22].

Among the deep learning topics discussed, only a select few are critically relevant to the objectives of this work. Let us focus the discussion exclusively on these essential aspects.

2.2.1 Training methods

There are many paradigms for leveraging data to train models, each with distinct methodologies and applications. For this work, the two that stand out are Supervised Learning and Self-Supervised Learning. Let's briefly discuss their values, differences, and situations where to use them.

Supervised learning

Supervised learning is a paradigm where the model is trained on a labeled dataset. This dataset comprises input-output pairs, with each input data point associated with a corresponding label or ground truth. The objective is for the model to learn a mapping from inputs to outputs that generalizes well to new, unseen data. Supervised learning tasks are typically divided into classification and regression, where classification refers to assigning inputs to discrete categories and regression to predicting continuous values.

Standard supervised learning is built upon the independently and iden-

tically distributed (i.i.d.) samples assumption. Independently here means that each sample is generated independently of the others. This implies that one sample's occurrence does not affect another's occurrence. The identically distributed suggest that all samples are drawn from the same probability distribution. This allows the model to learn from a consistent pattern or structure within the data and generalize well to new, unseen data from the same distribution. For deep learning in supervised settings, the i.i.d. assumption ensures that the model's performance on the training set is a reliable indicator of its performance on new data. Violations of this assumption can lead to overfitting[23], where the model performs well on the training data but poorly on unseen data because the training data is not representative of the overall distribution.

Self-Supervised learning

Self-supervised learning is an innovative paradigm where the model creates its own labels from the data itself, thereby transforming an unsupervised problem into a supervised one. The topic has gained prominence due to its ability to leverage vast amounts of unlabeled data to pre-train models. The learned representations can then be fine-tuned on smaller labeled datasets for specific downstream tasks. This paradigm is particularly impactful in domains such as natural language processing or even in deep tabular models[24][25].

2.2.2 Variable Types

As the scope of this work is towards relational databases, it is important to understand different variable types that can reside within a database. Also, to leverage the information value in the data, it is important to treat the variables according to their type. One of the possible partitioning of the variable types that will be used in this work is provided below. Importantly, this partitioning should not be confused with the SQL types (Sec. 2.1.2).

- **Categorical** - refers to a feature that can not be quantifiable.
  - **Nominal** - variables that represent distinct labels without any inherent order
  - **Ordinal** - represent categories with a meaningful order, such as

education levels (high school, bachelor's, master's, PhD)

- **Numeric** - variables that are quantitative.
  - **Discrete** - variables that take upon specific values, often integers, such as the number of children in a family.
  - **Continuous** - can take on any value within a range, like height or temperature.
  - **Cyclical** - can be discrete or continuous but with a cyclic range of values. A typical example can be the days of the week.
- **Plain Text** - consists of unstructured data in the form of strings, such as comments or reviews, requiring specialized processing to uncover the underlying information.

2.2.3 Attention Mechanism

Tables of the relational database can be viewed as sequences of attributes. When the values of the attributes are transformed into the embedding space, each row of the table can be seen as a sequence of embedding vectors of the attributes. The attention mechanism, as introduced by Vaswani et al. [18], enables the model to focus on relevant parts of the input sequence, enhancing the ability to capture dependencies and relationships between elements in a flexible and dynamic manner.

Attention

The attention mechanism can be described as a following succession of operations. First, the input vectors are linearly projected into queries (Q), keys (K), and values (V) using learnable weight matrices. This can be represented as:

$$Q = XW^Q, K = XW^K, V = XW^V \tag{2.1}$$

where W^Q, W^K and W^V are query, key and value weights.

Attention scores are then calculated as scaled dot-product, where the dot product of query and key is scaled by the dimension of the key matrix,

followed by a softmax operation to obtain the attention weights:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.2}$$

where d_k is the dimension of the key vectors. The result is a weighted sum of the values representing the attended information.

Multi-Head Attention

Multi-head attention extends the basic attention mechanism by employing n attention heads to jointly attend to information from different representation spaces at different positions of the input sequence. The input X is projected into n different query, key, and value spaces; this can be represented as:

$$Q_i = XW_i^Q, K_i = XW_i^K, V_i = XW_i^V \tag{2.3}$$

where W_i^Q, W_i^K, W_i^V are the weight matrices of the i-th head.

Each set of projections is used to compute scaled dot-product attention, resulting in n different attention outputs

$$head_i = Attention(Q_i, K_i, V_i) \tag{2.4}$$

The outputs from all attention heads are concatenated and linearly transformed to produce the final *multi-head attention* output

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_n)W^O \tag{2.5}$$

where W^O is the output projection matrix.

Self-Attention

Self-Attention refers to the situation when attention is applied to a single sequence, enabling the model to consider the relationships between all elements in the sequence. For an input sequence X with elements $x_1, \dots, x_n$, the self-attention mechanism computes the attention weights for each pair of elements within the sequence; the self-attention can be written as

$$SelfAttention(X) = Attention(XW^Q, XW^K, XW^V) \tag{2.6}$$

or with the use of multi-head attention as

$$SelfAttention(X) = MultiHead(X, X, X) \tag{2.7}$$

Cross-Attention

In contrast to self-attention, cross-attention involves two different sequences: the source and target sequences. This mechanism is crucial for passing information between two sequences and will be essential later in the work. In cross-attention, the queries Q are derived from the target sequence t, while the keys K and values V come from the source sequence s:

$$CrossAttention(Q_t, K_s, V_s) = Attention(Q_t W^Q, K_s W^K, V_s W^V) \tag{2.8}$$

or with the use of multi-head attention as

$$CrossAttention(Q_t, K_s, V_s) = MultiHead(Q_t, K_s, V_s) \tag{2.9}$$

2.2.4 Message-Passing

The relational databases contain not only the relations (tables) but also references between the primary and foreign keys (Sec. 2.1). These references suggest a graph-like structure will be needed to leverage the deep learning methods while keeping the connections established in the database.

GNNs can be used for deep learning on graphs with the use of the *message-passing* scheme. This paradigm is an iterative process that involves three primary phases: message creation, message aggregation, and node state update. To employ the GNNs in the learning process, it is important to understand this scheme.

Message Creation

In the message creation phase, each node sends messages to its neighboring nodes. This phase can be represented as

$$m_{u \to v}^{(t)} = MESSAGE^{(t)}(h_u^{(t-1)}, h_v^{(t-1)}, e_{uv}) \tag{2.10}$$

where $m_{u \to v}^{(t)}$ denotes the message sent from node u to node v at iteration t, $MESSAGE^{(t)}$ is a function that creates the message based on the state of the source node $h_u^{(t-1)}$, the state of the target node $h_v^{(t-1)}$, and possibly the edge features e_{uv}.

Message Aggregation

The message aggregation processes each node's messages received from its neighbors to a single value. This can be mathematically represented as

$$m_v^{(t)} = AGGREGATE^{(t)}(\{m_{u \to v}^{(t)} : u \in \mathcal{N}(v)\}) \tag{2.11}$$

where $m_v^{(t)}$ denotes the aggregated message for node v at iteration t and $AGGREGATE^{(t)}$ is a permutation-invariant function, such as sum, mean, or max, which combines the messages received from the neighboring nodes $\mathcal{N}(v)$.

Node State Update

Following the message aggregation, each node updates its state based on the aggregated message and its previous state. The update phase can be expressed as

$$h_v^{(t)} = UPDATE^{(t)}(h_v^{(t-1)}, m_v^{(t)}) \tag{2.12}$$

where $h_v^{(t)}$ is the updated state, $h_v^{(t-1)}$ is the previous state and $m_v^{(t)}$ is aggregated message of node v at iteration t.

These steps are repeated T times to allow for the propagation of the messages through the network. The actual functions used for the message creation, aggregation, and update differ between the types of GNNs and can greatly influence the capabilities of the model.

Chapter 3

Related fields of study

As the name suggests, this chapter discusses topics related to deep learning for relational databases. First and foremost, the chapter dives into the learning methods for tabular data as the topic can be viewed as a simplified case of the relational database with only a single relation (Sec. 2.1).

Further, the discussion will diverge towards the fields that can leverage the information provided by the references between the data, creating a graph or a network of some sort. These include fields such as Statistical Relational Learning (SRL), Neuro-symbolic integration, and, importantly, Graph Neural Networks (GNNs). The gap between tabular data learning and relational methods can be filled by a popular method called Propositionalization, which will also be discussed.

3.1 Learning from Tabular Data

Tabular data refers to a structured format for organizing and presenting data in tables, where information is arranged in rows and columns. In a table, each row typically represents a different record or data point, while each column represents a specific variable or feature of the data. Comparing this simple definition with definitions for the relational databases from Section 2.1, it is easy to see the connection between the two areas of study.

Methods that aim to use deep learning techniques on tabular data -

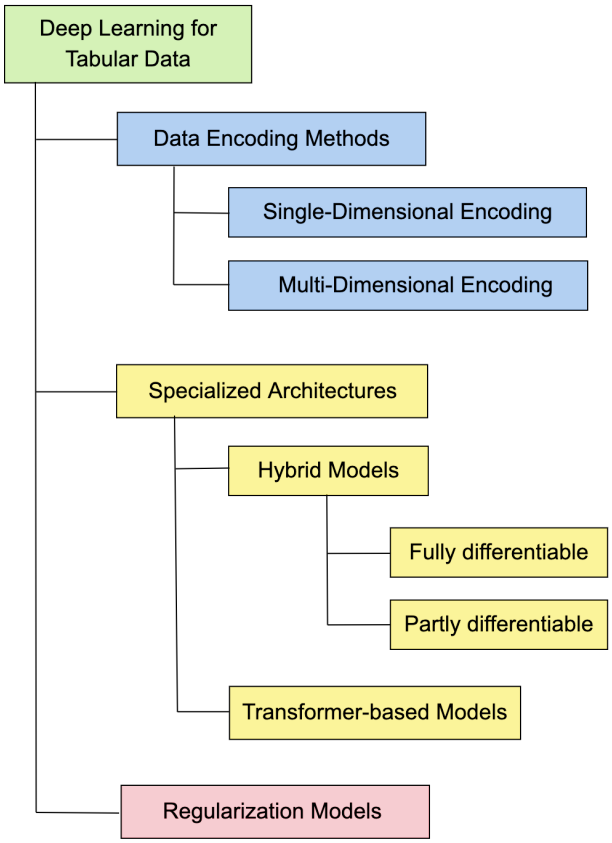


Figure 3.1: Unified taxonomy of deep neural network models for heterogeneous tabular data. [26, Borisov et. al., 2022]

Tabular Neural Networks (TNNs) - have shown some promising results in the last years, but standard techniques like rule learning and gradient-boosted decision trees[27] still dominate the analysis of tabular data [28][29]. Although it is not entirely clear why deep learning was not yet as successful on tabular data as it is on images[30], text[19] or even both at once, such as in multi-modal models[31], some possible reasons might be data quality[32], the difficulty of processing heterogeneous data[33] or disproportional importance of features[34].

Nevertheless, the taxonomy provided by *Borisov et al.* [26], as depicted in the Figure 3.1, offers a comprehensive view of the current approaches in deep learning for tabular data and can help the reader to orient themself

in the broad area of the tabular deep learning models. This is very useful for the purposes of this work as it is essential to familiarize oneself with the methods used on tabular data before extending the studied area to relational databases.

Borisov et al. [26] divides the deep tabular models into three base categories that further branch into several subcategories. The next few sections briefly discuss the most relevant areas from the taxonomy as described by *Borisov et al.*

3.1.1 Data Encoding Methods

The methods in the Data Encoding group focus on transforming categorical and numerical data to help deep neural network models extract information more effectively; techniques of this group are helpful for pre-processing or the initial layer of the DNN. These techniques can be divided into Single-Dimensional encoding methods that transform each feature separately and Multi-Dimensional encoding methods that map entire records to new representations.

Single-dimensional encoding methods are relatively simple approaches. In this group, there is, for example, Label Encoding, which maps each category to a numeric value, which can introduce artificial order; One-Hot Encoding, which adds a new binary column for each category, which can lead to high-dimensional sparse matrices or Hash-Based Encoding that uses hash functions to map categories to fixed-size values.

Multi-dimensional encoding methods are rather more sophisticated. To mention a few, there is the VIME [35] method that uses a *self-supervised* (Sec. 4.5.5) approach to learn the encoding of the records via detection of the corrupted cells. Other interesting encoding methods are SuperTML [36] and IGTD [37] that encode the table rows into images that can be then processed by CNNs.

3.1.2 Transfomer-based Models

The transformers[18] play an essential role in modern sequence processing models where the *attention* mechanism (Sec. 2.2.3) allows to focus on the

relevant parts of the input sequence. Generally, the transformer-based models can be split into two subcategories by the data that form a single record in the dataset - Table-level models, utilizing a pre-trained transformer model such as BERT[20] for processing a batch consisting of multiple tables, and Row-level models that provide a transformer-based architecture suitable for processing batches of rows usually from one large table.

Table-level transformers

The survey by *Badaro et al.* [38] provides an extensive overview of Table-level transformers. These models use language transformers to create embeddings of the table data without the need for knowledge about the variable types inside. As such, they are commonly used on large corpora of small tables such as WikiTables[39] where the training task is to predict the value of missing cells. The differences between them are primarily in the level of structure utilization. For example, the TaBERT[40] and the TaPas[41] models linearize the tables to create a long sequence of embedding vectors where positional encodings artificially add the tabular structure. TABBIE[42] creates separate embedding sequences for the rows and for the columns, each with its own positional encodings to provide information about the order in the row and the column. Column and row embeddings are then combined to form a single embedding vector for each cell that theoretically has information about the coordinates of the original cell. TUTA[43] introduces a concept of tree-based positional embeddings. These embeddings capture the cell's position using a bi-dimensional coordinate tree.

Row-level transformers

Although Table-level transformer model architectures are a promising topic in the deep learning field, the Row-level transformer models are more closely related to deep learning for relational databases as they focus on utilizing the information about the variable types (Sec. 2.2.2) of the columns and, as such, they are suitable for more complicated and precise tasks on larger tables where the training batch consists of rows. For learning from relational databases, these models can be instantiated for each relation in the database to process information within. Some notable examples of models in this category are TabNet[25] that uses custom modified transformer based archi-

tecture, TabTransfomer[44] focusing on the categorical values utilizing the original transformer encoder structure, SAINT[24] that introduces the Inter-sample Attention operation and Excelfomer[45] that focuses on the numerical attributes and leveraging pre-processing via CatBoost[46].

3.2 Relational Learning

As this thesis aims to provide a system that learns from relational databases, it is certain that relational learning is a related field of study. Relational learning is an area of machine learning (ML) that focuses on relational data, often represented in the form of graphs. This field has gained significant attention due to its ability to model complex relationships and interdependencies in data. Graph Neural Networks (GNNs) are at the forefront of this approach[22], but the field encompasses more subtopics that utilize relational data.

3.2.1 Statistical Relational Learning

Statistical Relational Learning (SRL)[47] is an area in machine learning that is closely related to First-Order Logic (FOL)[48] and Inductive Logic Programming (ILP)[49]. FOL provides a formal framework for representing and reasoning about relationships between entities. ILP uses FOL to learn interpretable models from structured data, representing knowledge as logical rules. SRL builds on these foundations by incorporating probabilistic elements into the logical framework, therefore enabling the modeling of uncertainty. One of the techniques that can be listed as an SRL method is RDN-Boost[50], which learns a series of function approximations using gradient boosting. The major drawbacks of SRL are hidden within its strengths, as it typically does not scale well due to foundations in FOL.

3.2.2 Propositionalization

Propositionalization[1][51] is a process or data transformation technique used to simplify complex relational data into a flat, propositional (tabular) format.

[1]Propositionalization can be viewed as a hyponym of feature engineering.

This process involves converting data that might be stored in multiple related tables (or other relational data) into a single table, where the goal is to make the data compatible with the tabular data learning techniques mentioned above. With this definition in mind, it is obvious that information is often lost during this process, which is undesirable and can lead us to sub-optimal models. That said, the propositionalization methods have dominated the industry in processing the data from relational databases[52]. Combined with gradient-boosted decision tree models such as XGBoost[27], they proved useful in gathering valuable information contained within the databases.

3.2.3 Neuro-Symbolic Integration

An intriguing area at the intersection of relational representations and deep learning principles is known as Neural-Symbolic Integration. A few neuro-symbolic frameworks operate with subsets of First-Order Logic (FOL) representations, such as Lifted Relational Networks[53], while integrating neural network principles. These methods theoretically enable deep learning from relational databases. However, similarly to SRL (Sec. 3.2.1), none of these methods scale effectively to real-world database sizes due to the inherent complexity of the FOL foundations, except for those that employ some form of propositionalization (Sec. 3.2.2) scheme[54].

3.2.4 Graph Neural Networks

Probably the closest field of study to deep learning for relational databases is Graph Neural Networks[55]. GNNs extend traditional neural networks to handle the complexities of graph data. Unlike datasets used in conventional neural networks, which assume i.i.d. samples (Sec. 2.2.1), graph data encapsulates relationships and interactions among data points. GNNs can be used for deep learning on graphs with the use of the *message-passing* (Sec. 2.2.4) scheme. This makes GNNs particularly effective for tasks where the connections between entities are crucial for understanding the underlying patterns.

Traditional GNNs assume a single type of node and a single type of relationship between the nodes. This simple scenario is not enough to model a relational database, but some extensions of GNNs come closer.

Multi-relational GNN

Multi-relational GNNs[56] extend the concept of GNNs by incorporating multiple types of edges. This allows the network to learn different kinds of relationships in a graph and how they uniquely contribute to the overall structure and function of the graph.

Heterogeneous GNN

Heterogeneous GNNs extend even further as they are designed to handle heterogeneous graphs, which contain different types of nodes and edges. This heterogeneity necessitates specialized handling due to the diversity in node and edge attributes; hence a specialized *message-passing* (Sec. 2.2.4) scheme is needed with a unique $MESSAGE$ function for each edge type and possible unique $AGGREGATE$ and $UPDATE$ functions for each node type.

Heterogeneous graphs allow one to model the high-level structure of relational databases where each pair of *primary* and *foreign* keys can be viewed as a unique edge type and tuples of different relations as having different node types. Yet, the heterogeneous graphs do not express the connections created between the attributes by the relation itself.

Hypergraph neural networks

On the other hand, hypergraphs generalize the concept of edges by allowing edges to connect any number of nodes; these general edges are called hyperedges. This capability enables hypergraphs to model more complex relationships and higher-order interactions among data points. Hypergraph Neural Networks[57] utilizes this generalization with extended *message-passing* (Sec. 2.2.4) scheme where $MESSAGE$ function is changed to operate on a set of nodes rather than a simple pair.

The relations (tables) of the relational databases (Sec. 2.1) can each be viewed as a single hyperedge connecting all attributes from the relation; hence, the tuples can one relation be thought of as nodes of the same type in this setting. Utilizing hyperedges will be important while modeling the structure of the relational databases.

3.2.5 Beyond Graph Neural Networks

There have been only very few works[58] in the past that focus on deep learning directly on relational databases. Notable work of ATJ-Net[59] mentions the idea of viewing relational databases as heterogeneous graphs, hence as an extension to GNNs, and also pinpoints the information loss issue during the propositionalization.

Fey et al. in Relational Deep Learning[60] present similar concepts of message passing, schema detection, and data retrieval. Fey et al. also propose an extension for the multi-relation hypergraph in the form of time information for nodes and the hyperedges, hence creating a temporal multi-relation hypergraph with an adequate message-passing scheme.

Chapter 4

Deep Learning for Relational Databases

In this chapter, I will venture into the field of tabular heterogeneous graphs. Starting with the representation of relational databases as heterogeneous hypergraphs (Sec. 4.1.1) and how to transform them into a form viable for deep learning methods. Further, I will discuss how to leverage the knowledge gathered from GNNs and tabular models, merging the ideas from both fields to enable deep learning from relational databases.

The result of these efforts is a proposal for a general blueprint model that follows the Graph Neural Networks's *message-passing* (Sec. 2.2.4) paradigm. Instances of the blueprint can encompass a wide range of possible models to allow for the utilization of different architectures proposed for tabular data, as well as the option to use various graph convolution layers. The blueprint, together with additional utilities for the transformation of data from RDBMS (Sec. 2.1.1) into graph datasets, forms a framework for learning from relational databases. Utilities include schema autodetection, transformation into a heterogeneous graph with provided schema, and storage and loading of the generated graph as a dataset for reusability without needing a database connection.

4.1 Data Representation

This work aims to provide a comprehensive end-to-end system for learning from relational databases. To accommodate such goals, the system seeks to

work directly RDBMS without the need to make any preprocessing steps outside the system. With this in mind, it would be best to learn directly from the raw data from the database. Although the relational databases based on the relational model have a predefined structure, the structure does not preserve the semantics of the columns necessary for the deep learning methods.

4.1.1 Heterogeneous Tabular Graph

To keep the structure defined by the relational model and consider work done in fields of Graph Neural Networks (Sec. 3.2.4) and Tabular learning models (Sec. 3.1), database data are represented as a *two-level bi-directional heterogeneous hypergraph*.

As a first level, there are tables, defined as the relation R between the values of attributes, forming n-ary tuples (Sec. 2.1). Each relation R_i can be viewed as *hyperedge* (Sec. 3.2.4) between all attributes from a single relation R_i due to the fact that the attributes are connected by being in the same relation R_i. Hence, a tuple t_j from relation R_i is an edge of hyperedge e_i. The graph will contain only k different hyperedges each for corresponding relation R_i where all tuples of relation R_i have the same structure in the form of attributes. The truth that each hyperedge is formed by attributes from a single relation allows for simplification that a *hyperedge* can be represented as a *node type*. This simplification is very important as it greatly reduces the complexity of the graph data.

On the second level, there are the primary and foreign keys (Sec. 2.1). Each pair of referenced relation R_1 and referencing relation R_2, where R_1 with primary key PK and R_2 with foreign key $FK(R_1)$, form a set of edges between the referenced and referencing n-ary tuples of appropriate relations. Each such pair can be represented as an edge type with a predefined source and target node type. It should be mentioned that each such edge type formed by primary and foreign key pair is also considered in a reversed direction, therefore making a bidirectional graph.

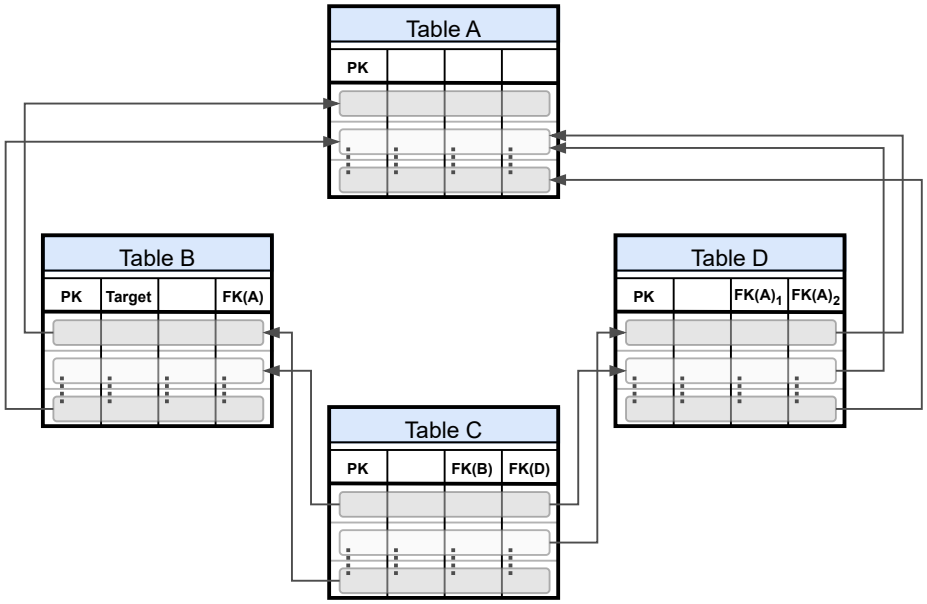


Figure 4.1: Representation of a hypothetical tabular graph with four tables. Table A can be viewed as the central table. Table B contains a target column and has a foreign key referencing table A. Table C has two foreign keys referencing tables B and D. Lastly, table D has two foreign keys, both referencing table A.

4.1.2 Schema Detection

The raw database values use SQL data types, such as 'text,' 'varchar,' or 'int' (Sec. 2.1.2). To use these values inside the deep learning methods, it is necessary to encode them in a suitable fashion that encapsulates the semantics of the variable type (Sec. 2.2.2). For example, the 'int' SQL data type can be a discrete numeric variable, an ordinal variable, or even a nominal variable. Similarly, the 'text' or 'varchar' SQL data type can represent plain text, a nominal variable, or also an ordinal variable in some cases.

Automatic schema detection is required to allow for a streamlined process that does not depend on user input. The automatic detection procedure is based on a series of straightforward rules and heuristics. To avoid overcomplicating this task, the procedure does not distinguish between ordinal and nominal variables (Sec. 2.2.2) but merely sees both as categorical variables. Other variable types that the procedure detects are numerical type, text type, and *cyclic* types, such as a date type and time type. On top of these types,

the procedure also copies the information about the primary and foreign keys with an auxiliary heuristic, which decides whether the given key is used only as a reference or if it is also a variable carrying additional information.

The output of the schema autodetection procedure is a JSON file. JSON is a format of choice for human readability[61] for structured data. This is important as the user can edit the output schema if required. The reason for such action might be a mistake in the original database model, missing foreign key definitions, or some expert knowledge about the variable types that the procedure could not catch.

4.1.3 Data Loading

With the relational database structure described as a graph (Sec. 4.1.1) and the data schema detected (Sec. 4.1.2), one can start the processing pipeline. At the beginning, there is the data loading. There are three options for how to load the database data to a graph that will be used for learning weights inside a DNN that this work considers.

- **Online** - Using directly the database located on a cloud server and transforming the data from the database to batches on the fly during the training process (possibly caching the generated batches in memory).
- **Local-online** - Coping the database to a local machine and transforming the data from the database to batches on the fly during the training process (possibly caching the generated batches in memory).
- **In memory** - Fetching the data from the database, proceeding with all necessary transformations, and storing the prepared graph on a local machine. During training, the whole graph is stored in memory.

The 'Online' and 'Local-online' options are mostly suitable for immense datasets that would not fit into the machine's memory. Still, on modern systems, the limitation for the size of the loaded dataset usually comes from the GPU memory. Moreover, the 'Online' and even the 'Local-online' alternatives are vastly slower than the third option as the transformations have to be done beforehand, and communication time with the database is also not negligible. For these reasons, most cases favor the 'In memory' option; nevertheless, the other options are present if necessary. Further focus will be put mostly on the 'In memory' case.

4.1.4 Graph Construction

To allow deep learning on graph data, it is necessary to construct the graph in a form that is compatible with such methods. The implementation leverages PyTorch Geometric (PyG)[62] and PyTorch Frame (PyFrame)[63] libraries to allow for future compatibility and ease of use as both libraries are built upon PyTorch[64], a greatly popular deep learning library. The PyG library provides a graph model with an implementation of the *message-passing* scheme (Sec. 2.2.4) necessary for the deep learning on graphs. The PyFrame library delivers a representation of tabular data.

Before creating a graph, it is important to encode non-numeric values into a numeric representation according to the variable type given by the schema (Sec. 4.1.2). DateTime values to integer *timestamp*, *categorical* labels to integer labels counted from zero, and notably *text* values can be encoded with a pre-trained language model such as Sentence-BERT[65] into a single embedding vector.

Having the data in numerical form, one can proceed with the graph construction. Here, the columns of each table (skipping the primary and foreign keys used only for referencing) are concatenated into separate TensorFrame[63] forming an aforementioned hyperedge (Sec. 4.1.1). By joining the inter-referencing table pairs, edge indexes are created. Edge indexes form two arrays where on the same position in one array is an index of the referenced row, and in the second is an index of the referencing row. These parts are combined into HeteroData[62] to form the Heterogeneous Tabular Graph.

4.1.5 Data Sampling

The graph described in Section 4.1.1 and constructed in Section 4.1.4 can be *connected*. Connectivity plays an important role in graph-related optimization problems such as the Traveling Salesman Problem[66]. In the case of applying deep learning methods, it strikes as a limiting factor.

In deep learning, training of the model is typically done from batches constituting of n tuples (x_i, y_i) that represent just a fraction of data points of the whole dataset and are sampled with the i.i.d. assumption on the underlying random variables (Sec. 2.2.1). In the context of the relational model, the simplest valuable form of y_i to be considered for supervised

learning is the value of a single attribute A_{target} of a target relation R_{target}. For training purposes, the target attribute is *removed* from the graph. As for the x_i values, two main cases can be assessed.

- Each row r_i of the target table $T_{R_{target}}$ forms, through references to other tables, a graph that is disjunctive to all other graphs formed by rows r_j where $i \neq j$ (that is apart from hyperedges of the relations).
- Any row r_i of the target table $T_{R_{target}}$ can be related to row r_j of target table through k edges; hence graph induced from BFS [67] starting at r_i can possibly span through whole dataset graph.

Typically, the first case proves to be uncommon; therefore, it will not be considered. The second case is also more general, making it ideal for the set goal of the universal pipeline. Nevertheless, the x_i examples can not be considered as i.i.d. tuples as opposed to the standard supervised learning.

A simple resolution to the connected graph issue is to have only a single learning sample - the whole dataset graph. Unfortunately, this solution is only feasible for smaller datasets. For larger datasets, one has to resort to sampling batches of sub-graphs.

A naive option is to use the aforementioned BFS. This method is used for the 'Online' methods mentioned in Section 4.1.3 as using any advanced approach proves to be challenging, to say the least. The BFS runs natively on the database via a series of recursive SQL joins to a limited depth from the sampled rows of the target table. This procedure results in a sub-graph that can be processed as described in Section 4.1.4 and used for learning objectives.

As for the 'In Memory' case, two approaches have been tested for sampling the mini-batches. First is a 'Neighborhood Sampler' introduced in *"Inductive Representation Learning on Large Graphs"* [68]. Neighborhood Sampler works very similarly to BFS with an extension for limiting the depth and the number of edges used for each node (optionally, different limits on the number of edges can be applied at each depth and for each edge type). The second considered alternative is HGSampling described in *"Heterogeneous Graph Transformer"* [69]. HGSampling is a considerably more sophisticated method with roots in BFS. The key difference to Neighborhood Sampler is that it limits the number of nodes of a given type rather than the number of edges. HGSampling also introduces a concept called 'node budget,' which

determines the probability of selecting a node of a particular type during sampling. Through thorough testing, it was assessed that the HGSampling would be the main sampling method as it proved to be more resilient to the exponential growth of the sampled sub-graph.

4.2 The Blueprint

To combine the knowledge of previous sections and the main goal of this thesis, the blueprint proposes a general model for Deep Learning on Relational Databases. An important difference to traditional GNN models is that the blueprint is built upon the tabular nature of the relational database data, hence keeping the dimensionality of the rows and columns. That being said, the blueprint allows for the instantiation of models that do not strictly follow this pattern, and because of that, the blueprint can be viewed as an extension of classical GNN models.

The Blueprint consists of several components representing general *differentiable* functions with predefined input and output structures. These general functions can be split into three categories. [1]

- **Transformation** - a function with a single input and single output, described as $T : X \overset{1:1}{\rightarrow} Y$, where $rank(X) \leq rank(Y)$. Transformations can occure as tuple *transformations* $T_t : t \overset{1:1}{\rightarrow} t'$ or as attribute *transformation* $T_a : a \overset{1:1}{\rightarrow} a'$.
- **Combination** - a function that takes a fixed number of inputs and *combines* them into a single output; $C : (X_1, X_2, \dots, X_N) \overset{N:1}{\rightarrow} Y$, where $rank(X_i) = rank(Y)$. Combinations can be further divided into two sub-categories, attribute *combinations* $C_a : (a_1, \dots, a_n) \overset{N:1}{\rightarrow} (a')$ and tuple *combinations* $C_t : (t_1, \dots, t_n) \overset{N:1}{\rightarrow} (t')$.
- **Aggregation** - a function with permutation invariant set of inputs $A : \{X_1, X_2, \dots, X_M\} \overset{M:1}{\rightarrow} Y$, where $rank(X_i) = rank(Y)$. Aggregations

[1] The definitions contain a $rank$ function that symbolizes the tensor *rank*, sometimes also called *degree* or *order*, not to be mistaken with the rank of a matrix. It is not trivial to provide a definition for such a function, yet here is at least a partially formal attempt $rank(X) = m | X \in \mathbb{R}^{D_1 \times \cdots \times D_m}$.

considered by this work should always be on the tuple level; hence, the *aggregation* function can be denoted as $A_t : \{t_1, \dots, t_m\} \overset{M:1}{\rightarrow} t'$.

4.2.1 Modules

The Blueprint consists of several modules that are defined by the function categories described above (*transformation*, *combination*, *aggregation*). Here, I will discuss each section of the blueprint in close detail. For additional context, see the Figure 4.2. It should be noted that *each* of the following modules is *instantiated* for *each* table.

Embedder

Every instantiation of the blueprint starts with the Embedder module that takes attribute values encoded as described in Section 4.1.4 and transforms them into embeddings of the dimension D. Dimension D can theoretically have different values for each table or perhaps even sets of columns inside a single table. Still, for the sake of simplicity, only a single global value of D is considered. By the defined terminology, the Embedder module is a block of attribute *transformations* T_a. Each value of attribute a_j of relation R_i is converted by the function $T_{a_j}^{R_i} : a_j \rightarrow a'_j$, resulting to m tuples $(a'_1, \dots, a'_n) \in (\mathbb{R}^D, \dots, \mathbb{R}^D)$.

Post-Embedder

'Post-Embedder' is an optional module that allows for the application of a specific function after generating the embeddings. There are two cases for the structure of this module, either a tuple *transformation* T_t, e.g., *positional encoding* [18, 42], or attribute *combination* C_a, e.g. simple attribute concatenation or whole tabular model such as 'Trompt'[70].

Each of the following 'Pre-Combination,' 'Combination,' and 'Post-Combination' modules is repeated in N layers as depicted in the Figure 4.2.

Pre-Combination

The first of the repeating modules is the Pre-Combination. This module can be defined as a tuple *transformation* T_t and provides the major processing opportunity for gathering information about the given relation R_i, e.g., 'Self-Attention'(Sec. 2.2.3) or even whole 'Transformer Encoder'[18]. If necessary or requisite, Pre-Combination can also be skipped.

Combination

Next follows the Combination module, which consists of several sub-sections and evaluates the graph structure of provided data through the extended *message-passing* scheme (Sec. 2.2.4 and 3.2.4). Component starts with tuple *combination* $C_t : (t_i, t_j) \overset{2:1}{\rightarrow} t'_i$ with each $t_j \in R_2$ where $t_i[FK_{R_2}] = t_j[PK_{R_2}]$, resulting into a set of $\{t'_{i_{R_2}}\}$ representations for each such pair of $t_i \in R_1$ and the related R_2. Every such set $\{t'_{i_{R_2}}\}$ of m tuples goes through tuple *aggregation* $A_{t'} : \{t'_{i_{R_2}}\} \overset{m:1}{\rightarrow} t''_{i_{R_2}}$, resulting into a set of $\{t''_{i_{R_k}}\}$ tuples for each referenced relation R_k. Lastly, the set $\{t''_{i_{R_k}}\}$ of l tuples goes through second tuple *aggregation* $A_{t''} : \{t''_{i_{R_k}}\} \overset{l:1}{\rightarrow} t'''_{i_{R_1}}$. With that, the dimensionality of the output should correspond to the dimensionality of the input of the Combination module. For instance, the *cross-attention* (Sec. 2.2.3) or SAGEConv[68] can be used as the C_t and mean or sum for both *aggregation* functions.

Post-Combination

Lastly, the blueprint model can be *optionally* instantiated with the Post-Combination module that allows for residual connection between the input and output of the Combination block. More exactly the block can be defined as a tuple *combination* $C_t : (t_{in}, t_{out}) \overset{2:1}{\rightarrow} t'$. This module allows for iterative aggregation of results of each of the N layers within the blueprint.

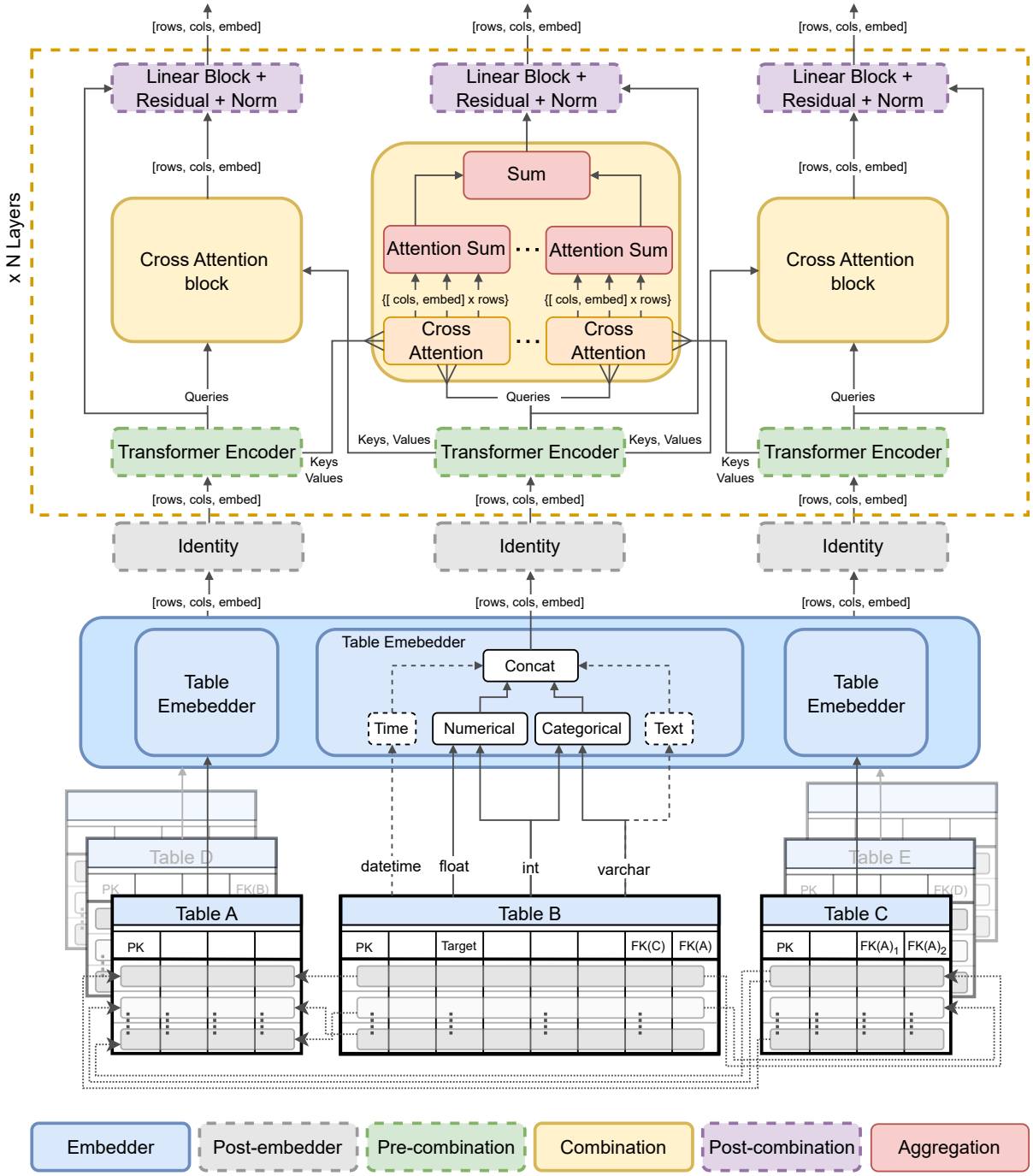


Figure 4.2: A schema of Database Transformer; instance of the blueprint model for Deep Learning on Relational Databases.

4.3 Blueprint Classes

With the blueprint summarized, one can proceed to discover the possibilities of the models inside this neural architecture space. Based on the dimensionality and type of the chosen components of the blueprint, the instances can be split into several categories.

4.3.1 Embedders

The initial processing of the data can greatly influence the effectiveness of the model. Coming from the work done in the field of tabular DL models, there is a variety of possible approaches. Nevertheless, the categorical variables are almost always encoded with simple embedding, with the exception of the models that do not use the *categorical* variables, e.g., Excelformer [45]. Here is a list of some of the options focusing on the other variable types.

- **Stack Embedder** - simplest option to increase the dimensionality of the *numeric* attributes is to copy the value D types in the embeddings vector, where D is the target dimension of the embeddings.
- **Linear Embedder** - linear layer with no *activation* function and one input channel and D output channels is also a common way to create the embedding vectors out of *numeric* variables.
- **Text Embeddings Transcoder** - as discussed in Section 4.1.4 plain text data from the database can be processed by the pre-trained language model. Because it is unlikely that the language model embedding dimension will match the dimension D set in the blueprint instance, a linear layer with no activation layer can be leveraged to address the dimensionality difference.
- **Timestamp Embedder** - the most sophisticated embedder listed here. Time-related attributes pose a challenge as they often encode periodical information, such as that on Friday, there will be, on average, fewer people at the university. If the day in the week is encoded as a categorical attribute, then this example is solved. Still, if the more complex periodical behavior is present, encoding it with just categorical or numerical attributes is extensively more complicated. To account for this kind

of possible periodical information that might be encapsulated by the year, month, day, etc., of the timestamp, the embedder first uses cyclic encoding [63] 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 classical tabular models usually take the opportunity only of combining simple embedding for the *categorical* variables and Stack Embedder or Linear Embedder for the *numerical* variables, but the usage of the text and timestamp attributes can lead to additional performance gain.

4.3.2 Inner Dimensionality

The chosen function type inside the Post-Embedder determines the dimensionality of data processed by subsequent parts of the blueprint instance. Based on whether the tuple *transformation* or attribute *combination* is chosen for the Post-Embedder, models can be split into two classes.

- **Reduced** - Upon selecting the attribute *combination* function, data dimensionality is reduced, effectively creating a single embedding vector for each tuple. In such case, the tuple before the Post-Embedder has a dimensionality of $\mathbb{R}^{A \times D}$, where A is a number of attributes and D is the embedding dimension. Output x of the Post Embedder has dimensionality $\mathbb{R}^X$ where $X \in \mathbb{N}$. A typical simple function that falls in this category is a *flatten* operation where the output x would have a dimensionality $\mathbb{R}^{A*D}$ with A being a number of attributes and D the embedding dimension of the original tuple. These models can be seen as standard Heterogeneous GNNs[71].

- **Tabular** - Skipping the Post Embedder or choosing the tuple *transformation* results in keeping the tabular structure of the data. Post Embedder transformation can be described as function $T_t : t_i \to t'_i$, where $t_i \in \mathbb{R}^{A \times D}$ and $t'_i \in \mathbb{R}^{A \times D'}$ with A being a number of attributes, D the embedding dimension and D' the transformed embeddings dimension.

4.3.3 Tabular Models

One of the goals of this work is to integrate the Tabular Models with the general blueprint for relational databases. To instantiate the blueprint with tabular models such as mentioned in the Section 3.1.1, there are two ways that can be characterized by the role that the model takes.

- **Tabular Encoder** - tabular model can be placed in the Post-Embedder module. In that case, the data will go through it once at the beginning of the model, and the rest of the instance modules can be chosen appropriately to the 'Tabular Encoder' output dimensionality, e.g., the 'Database Transformer' described in Section 4.4.
- **Tabular Graph Model** - the tabular model is placed in the Pre-Combination module; hence, it has to follow the condition of being a tuple *transformation*. These instances repeat the tabular model in each of the N layers of the blueprint together with the chosen Combination block.

4.4 Database Transformer

The blueprint model can be theoretically instantiated with any functions that fit the rules defined in Section 4.2. To highlight the interesting parallels between relational database data structures and those used in the *transformer* architecture, the Database Transformer is a proposed model that extends the *transformer*'s encoder and *cross-attention* mechanisms to the context of relational databases.

A single row of tabular data passed through the Embedder can be viewed as a sequence of *different* features encoded in the embeddings spaces of the respective attributes. This lurks the idea of using the transformer for processing the information embedded within the data as the transformers are meant for the evaluation of sequences of embedding vectors.

Database Transformer depicted in Figure 4.2 (or an alternative version of the schema in Figure B.1) takes advantage of the self-attention as well as the cross-attention functions. Each of the encoded tables from the relational database is first passed through the Embedder module to create the

embeddings. Then follows N repeating layers of the Pre-Combination, Combination, and Post-Combination modules as the Post-Embedder is skipped in the baseline (Sec. 4.5.1) version. Nevertheless, the Post-Embedder can also be utilized to add the *positional encoding* to the embeddings as tested in the further Section 5.3.5.

Each of the Pre-Combination modules inside the N repeating layers hosts the Transformer Encoder as originally described by *Vaswani et al.*[18]; hence a self-attention (Sec. 2.2.3) followed by a residual connection with a 'Layer Normalization'[72], continued by a *feedforward* Neural Network (FNN). Two linear layers with a ReLU activation function between them are used for the FNN followed again by a residual connection with a Layer Normalization. An important difference between the original Encoder by *Vaswani et al.* and the one used in the Database Transformer is that each Pre-Combination module contains only one Self-attention block and one FNN block. In contrast, these blocks are repeated M times in the original Transformer model.

The Combination and Post-Combination together form the Transformer Decoder part of the model. Tuple outputs of each table's T_{R_i} Pre-Combination module described in the previous paragraph are now processed by the *cross-attention*. The function of *cross-attention* operation done between the tables T_{R_1} and T_{R_2} that share a connection can be written as follows

$$C_t^{cross-attn}(t_i, t_j) = Attention(query = t_i, key = t_j, value = t_j) \tag{4.1}$$

The resulting set of tuples $\{t_{i_{R_2}}\}$ for table T_{R_1} is processed by *attention* sum operation and further by a simple sum function resolving to a processed tuple t_i of relation R_1.

The Post-Combination module starts with a residual connection that sums the outputs of the Pre-Combination and Combination modules, followed by a Layer Normalization. Finally, the Post-Combination module ends by passing data through another FNN block, the same as in the Pre-Combination module.

Finally, the Pre-Combination, Combination, and Post-Combination modules are repeated N times to allow information propagation through the model network.

4.5 Experimental Blueprint Instances

The blueprint allows for immense possible instantiations. To list a few, the following section describes instances further used for experiments (Sec. 5.3). Instances include a variety of Database Transformer versions as well as a set of instances based on tabular models (Sec. 3.1.2). This section provides a walkthrough of their structure and components.

4.5.1 Database Transformer Versions

The Database Transformer, as described in previous Section 4.4, consists of N layers, where each layer can be written in terminology of Section 4.2 as

$$C_t^{FFN+Norm} \circ A_t^{Sum} \circ A_t^{Attn} \circ C_t^{Cross-Attn} \circ T_t^{Transformer-Encoder} \tag{4.2}$$

The Database Transformer stands as the leading model of the thesis, and it was tested with an additional list of options, including different types of Embedder and Post-Embedder modules (Sec. 4.2.1 and 4.3.1).

- **Baseline** - Embedder uses only *categorical* and *numerical* variables with simple embeddings and Linear Embedder for respective *variable* types. This model will serve as the central comparative point to other blueprint models.
- **With Text** - Extends the baseline embedder with text embeddings created in the graph construction phase (Sec. 4.1.4) by Sentence-BERT [65]. To match the dimension of embedding vectors of other attributes, the initial text embeddings are transformed by the Text Embeddings Transcoder.
- **With Time** - Extends the baseline embedder with datetime attributes transformed by the Timestamp Embedder. Time-related attributes pose a challenge as they often encode periodical information, which is hard to grasp. Timestamp Embedder greatly helps with such tasks.
- **Positional Encoding** - Extends the baseline by adding a *positional encoding* function into the Post-Embedder module. If the database table consists of many attributes with similar information values, it might be beneficial to provide the transformer model with data about the order

of the attributes. This is based on the original concept of *positional encoding* inside the Transformer [18].

4.5.2 SAGE GNN

This model can be classified as 'Reduced' (Sec. 4.3.2) by its inner dimensionality because of the use of the Post-Embedder function that flattens the columns' dimension; $C_a : (a_1, \dots, a_n) \rightarrow (a_1 \dots a_n)$, where $(a_1, \dots, a_n) \in (\mathbb{R}^D, \dots, \mathbb{R}^D)$ and $(a_1 \dots a_n) \in \mathbb{R}^{n*D}$. Reduced dimensionality allows for the use of standard graph convolution functions; in this case, the SAGE[68] is utilized. The N repeating layers can be described as

$$A_t^{Sum} \circ A_t^{Sum} \circ C_t^{SAGEConv} \circ T_t^{BatchNorm+ReLU} \tag{4.3}$$

where the Post-Combination module is skipped. Embedder uses only *categorical* and *numerical* variables with simple embeddings and Linear Embedder as in the baseline DB Transformer.

4.5.3 SAINT

SAINT refers to the tabular model introduced in *"SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training"*[24]. SAINT takes a 'Transformer Encoder'[18] layer and extends it by a second block that uses 'Intersample Attention,' details of which can be found in the article.

The blueprint instance utilizes the 'SAINT Encoder' layer as the Pre-Combination module in a mixture with the *cross-attention* for the Combination module. The model also uses a comparable Embedder as the DB Transformer baseline with an extension that a ReLU activation function follows the linear transformation. The N repeated layers can be defined as

$$C_t^{FFN+Norm} \circ A_t^{Sum} \circ A_t^{Attn} \circ C_t^{Cross-Attn} \circ T_t^{SAINT-Encoder} \tag{4.4}$$

Notably, the architecture of the SAINT blueprint instance resembles that of the Database Transformer.

4.5.4 Trompt

The instance closely follows the Trompt[70] architecture as described in the article. The Trompt Encoder is used once at the beginning as the Post-Embedder module to transform the data. The N repeating layers have a simple definition as

$$A_t^{Sum} \circ A_t^{Sum} \circ C_t^{AddMean} \tag{4.5}$$

where the Pre-Combination and Post-Combination modules are skipped and

$$C_t^{AddMean}(t_i, t_j) = t_i + \frac{1}{dim(t_j)} \sum_{a_k \in t_j} a_k \tag{4.6}$$

Notably, the model utilizes the Trompt Decoder as a prediction head and has a custom Embedder that extends the DB Transformer baseline by following the categorical embeddings by Layer Normalization'[72] and the linear transformation of numerical values by a ReLU activation, and also a Layer Normalization.

4.5.5 TabNet

Another instance based on a tabular model is a TabNet[25]. TabNet is an *attention* based architecture that defines two main modules - encoder and decoder. The article describes the Feature Transformer and Attention Transformer blocks that are used to build the encoder and decoder modules. The encoder is formed by a series of repeated Feature Transformers, each followed by the Attention Transformer. The decoder, used mainly for *pre-training*, consists only of Feature Transformers.

Similar to the SAGE GNN, TabNet can be classified as 'Reduced' by its inner dimensionality. Embedder only processes the *categorical* variables through simple embeddings, and the *numerical* variables are duplicated to the target dimension by the Stack Embedder. The N repeating layers can be written as

$$A_t^{Sum} \circ A_t^{Sum} \circ C_t^{AddMean} \circ T_t^{TabNet-Encoder} \tag{4.7}$$

where the $C_t^{AddMean}$ is defined in equation (4.6).

4.5.6 TabTransformer

The last experimental blueprint instance is based on the TabTransformer[44]. The TabTransformer architecture primarily processes the *categorical* attributes as *numerical* attributes are passed just through a Layer Normalization. The *categorical* columns are passed through a Transformer Encoder block.

In the TabTransformer blueprint instance, a naive Embedder is used. Similarly to the TabNet instance, the Embedder uses simple embeddings and a Stack Embedder for *numerical* attributes to avoid transformations of the values. The rest of the N repeating layers are described as follows,

$$A_t^{Sum} \circ A_t^{Sum} \circ C_t^{AddMean} \circ T_t^{Transformer-Encoder/LayerNorm} \tag{4.8}$$

where the $C_t^{AddMean}$ is defined in equation (4.6).

Chapter 5

Experiments

This chapter presents the results of the conducted experiments. First of all, the discussion will turn to the datasets utilized for the evaluation of the models and, importantly, the employment of the CTU Prague Relational Learning Repository. Next, the experimental pipeline will be described together with the technologies involved. Notably, it goes through experiment runs as well as the hyperparameters used and other specifics of the experimental environment. Emphasis will be put on the description of the learning process used for experiment tests.

5.1 Benchmark Datasets

To effectively evaluate progress in a given field across the publications, it is appropriate to use reproducible experimental settings with publicly available resources that test the accuracy of a given model. Reasons like these lead to the existence of numerous benchmarking datasets often hosted on websites with scoreboards.

One of the first such projects was the *"UCI Machine Learning Repository"*[73] created by David Aha in 1987, focusing mostly on tabular data. Also, sites like 'Papers with Code'[1] and Kaggle[2] aggregate the datasets and provide a public score of submitted models.

[1] `https://paperswithcode.com/dataset`
[2] `https://www.kaggle.com`

5.1.1 Existing Benchmark Datasets

There are many publicly available datasets for tabular data on the aforementioned sites. Datasets such as WikiTables[39] or MIMIC[74] are well-known and commonly used to assess the performance of a tabular model. Graph or even Heterogeneous Graph datasets can also be easily accessed. SNAP[75] provides an extensive collection of network graph datasets, the 'Open Graph Benchmark'[76] project directly aims at comparing the performance of GNN models on large graphs.

Recently, in the work of RelBench[60] project, an effort was made to create a benchmark that targets data of relational databases. The RelBench project contains two large datasets; 'Stack Exchange' and 'Amazon.' Sadly, RelBench shares a similar problem to the graph datasets for purposes of this work. The datasets are not stored inside a relational database but rather in the simplified form of CSV files annotated with additional data about the primary and foreign keys. Not storing the data inside the relational database represents an issue, as the goal of this thesis is to provide an end-to-end workflow directly from the relational databases.

5.1.2 The CTU Prague Relational Learning Repository

Luckily, another project at CTU solves the issue of a collection of relational databases. 'The CTU Prague Relational Learning Repository'[77] (CTU Relational), originally created by Jan Motl in 2015, contains more than 50 relational databases stored on the MariaDB[3] server. The repository was used for machine learning tasks in the past and stores the scores of selected models.

As of the end of 2023, the server was discontinued on the original address `relational.fit.cvut.cz`. To preserve valuable historic progress data and the SQL databases aggregated on the server, it was decided to make administration of the repository part of this project. The new database server of the CTU Relational is available at `https://relational.fel.cvut.cz`.

[3] `https://mariadb.com`

Repository Migration

As the CTU Prague Relational Learning Repository was included in the scope of this thesis, it is only logical to discuss the migration process from the discontinued server to the new one. The original admin of the CTU Relational provided per-database dumps from the old MariaDB server. A dump is a file used for backup or transfer to another database server (not necessarily MariaDB or MySQL).[4] To ensure better environment stability and clarity, the new MariaDB server is run inside a Docker container[5]. Part of this repository was also a web server that communicates with the SQL server to present parameters and schema visualization of stored databases and also to showcase the performance of different models on the datasets. Both servers are wrapped in a single Docker compose file with a simple network that allows the web server to communicate with the SQL server locally. A migration script needs to be executed to provide the SQL servers with the actual data. This script downloads old database dumps, creates the databases, and copies the data directly to the MariaDB server.

Database	#Tables	#Instances	Size in MB	Task
Accidents	4	495760	210.0	Class
AdventureWorks	71	30669	234.6	Regr.
AustralianFootball	4	3036	38.0	Class
BasketballMen	9	1536	18.2	Regr.
Biodegradability	5	328	3.2	Regr.
Carcinogenesis	6	329	26.3	Class
CCS	6	1000	658.4	Regr.
ClassicModels	8	273	0.5	Regr.
Countries	4	247	8.6	Regr.
Credit	9	10084	443.6	Class
CS	8	100	0.3	Class
Dunur	20	276	0.8	Class
Elti	14	1081	0.7	Class
Employee	7	2838426	344.6	Regr.
Financial	8	682	94.1	Class
FTP	2	29555	7.5	Class
Genes	3	862	1.9	Class
Hepatitis	7	500	2.2	Class

[4] https://mariadb.com/kb/en/mariadb-dump/
[5] https://www.docker.com/resources/what-container/

Database	#Tables	#Instances	Size in MB	Task
Hockey	23	7759	15.5	Class
IMDb	7	794625	614.6	Class
MovieLens	7	6039	151.9	Class
Lahman	25	23111	84.0	Regr.
LegalActs	5	564268	238.2	Class
Mesh	32	223	1.1	Regr.
Mondial	33	454	3.3	Class
MooneyFamily	72	92	3.3	Class
Mutagenesis	3	188	0.9	Class
Nations	3	14	2.1	Class
NBA	4	30	0.3	Class
NCAA	10	268	40.6	Class
Northwind	29	830	1.1	Regr.
Pima	14	768	0.8	Class
PremierLeague	4	363	11.3	Class
PTC	4	343	7.8	Class
PTE	41	299	7.3	Class
Pubs	11	18	0.4	Regr.
Sakila	16	15991	6.6	Regr.
SalesDB	4	6148886	539.3	Regr.
SameGen	7	1081	0.3	Class
Stats	8	38357	621.4	Regr.
StudentLoan	13	1000	0.9	Class
Thrombosis	3	806	1.9	Class
TPCC	9	28433	174.1	Class
TPCDS	24	99550	4587.5	Class
TPCH	8	148255	1925.1	Regr.
Trains	2	20	0.1	Class
University	5	38	0.3	Class
UW-CSE	4	278	0.2	Class
VOC	8	8215	2.7	Class
World	3	239	0.8	Class

Table 5.1: List of databases from The CTU Prague Relational Learning Repository with introductory description.

5.2 Hyperparameter Optimization

In the realm of machine learning and deep learning, the performance of a model is often determined by a multitude of factors, including the architecture of the model, the choice of optimization algorithm, and the quality of the training data. However, one crucial aspect that can significantly impact a model's effectiveness is often overlooked: hyperparameters. Hyperparameters are the configuration settings that govern the training process, such as learning rates, batch sizes, and the number of hidden layers in a neural network. Selecting the right hyperparameters can be a challenging and time-consuming task, as their optimal values can vary depending on the specific dataset.

Hyperparameter optimization is the systematic process of finding the best set of hyperparameters for a machine learning model, with the goal of maximizing its performance and generalization ability. A more robust and mathematically driven problem statement can be found in the article [78]. Nevertheless, the significance of hyperparameter optimization cannot be overstated, as it can mean the difference between a mediocre model and a state-of-the-art one.

As previously stated, hyperparameter optimization is a difficult task that comes with a lot of challenges. Here is a list of just a few: high dimensionality, computational and time extensive, complex interactions, automated tuning, and parallelization. In addressing the intricate challenges of hyperparameter optimization, particularly those linked to automated tuning and parallelization, it is prudent to utilize specialized tools such as Optuna[79], Ray[80] or MLFlow[6].

5.2.1 Distributed Computing

Parallelization on server clusters addresses the computationally intensive nature of hyperparameter optimization. Enabling the hyperparameter optimization process to be executed concurrently across multiple processing units is a complicated task that requires an effective and robust management system to ensure smooth operation. With such a system, parallelization significantly curtails the time and computational resources required, thereby streamlining

[6] `https://mlflow.org`

the optimization process. One of the such systems for the management of distributed computing tasks is Ray[80].

5.2.2 Tuning

Automated tuning is a pivotal aspect of this hyperparameter optimization. The intelligent automated hyperparameter selection process is necessary for the exploration of a broad search space. It is not feasible to merely traverse search spaces consisting even of a few parameters; hence, there is a necessity for both intelligent and resource-conscious approaches to solve the search task. An example of such a system can be the Optuna[79].

5.2.3 Aggregation

The Hyperparameter optimization tasks are usually composed of many experiments running over long periods while generating plenty of valuable metrics; hence, it is important to log, persist, and aggregate the produced pieces of information. As such assignment is not trivial, using the dedicated system to manage the experiments' storage is often valuable. This integration ensures not only a systematic approach to recording results but also facilitates comparative analysis and effective management of the machine learning workflow. MLFlow can be listed as an example of such a system.

5.3 Experiment Runs

At last, this section will talk about the process of using the blueprint instances (as defined in Section 4.5) to accomplish various machine learning tasks. Briefly mentioned are also the environment and parametrization of the models and training. Further, the overall results and the effects of alternative modifications of the Database Transformer model will also be examined.

5.3.1 Tasks

The models are evaluated on two supervised training tasks, that is, *classification* and *regression* (Sec. 2.2.1). Out of all 49 datasets in the CTU Relational repository, as listed in the table 5.1, only 19 classification and 16 regression datasets were selected. Most of the discarded datasets were removed because the prediction task was considered too trivial to achieve. Some of the databases were also too small and, hence, impractical to be used for assessing the quality of the model.

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	28027	64122	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	21497	43030	True	True
MuskSmall	2	1	1	5.17	568	476	False	False
mutagen.	3	3	4	26.03	10324	15379	False	False
Pima	9	8	1	8	6912	6144	False	False
Prem.Leag.	4	5	3	29.29	11308	31867	True	True
Toxicology	4	5	1	53.26	49813	92541	False	False
UW_std	4	4	4	1.49	712	604	False	False
WebKP	3	3	1	94.16	81850	82581	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	29125	20994	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	4.2	5.6M	8.2M	True	False
tpcd	8	10	5	11	8.7M	27.2M	True	True

Table 5.2: List of classification datasets used in experiments with statistics.

The selected datasets for the classification and regression tasks can be viewed in the tables 5.2 and 5.3, respectively. Tables contain statistics about the relational databases that they represent. 'Num. Rels.' - number of relations inside the 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' - overall number of rows in all tables of the database, 'Total Num. Edges' - overall number of primary, foreign key pairs between all tables of the database, 'Text Col.' - whether the database contains non-key text attribute, 'Time Col.' - whether the database contains *datetime* attribute.

To use the blueprint models for prediction tasks, the output of the *last* layer produced for the *target table* is flattened (if necessary) and finally processed by MLP prediction head with M layers where each of the hidden layers is optionally followed by 'Batch Normalization'[81] and by ReLU activation function. To enable gradient descent, in the classification jobs, the MLP output is put through the *cross-entropy*[82] loss function or through MSE loss function in case of the regression jobs.

As for the metrics used for the results, an accuracy metric can be leveraged for the classification as it is a general, easy-to-understand value without the need for a deeper understanding of the data and the actual prediction task. Accuracy cannot be employed on regression. To have a somewhat comparable metric across the datasets, even for the regression tasks, a 'Normalized Root Mean Squared Error' (NRMSE) was used. The $NRMSE$ function is defined as

$$NRMSE(y, \hat{y}) = \frac{RMSE(y, \hat{y})}{\bar{y}} \tag{5.1}$$

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

$$RMSE(y, \hat{y}) = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y})^2}{n}} \tag{5.2}$$

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	21895	33094	False	False
classicmod.	8	7	2	1	3864	6846	True	True
GOSales	5	4	1	39.5	151105	188759	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	44822	62744	True	True
restbase	3	3	2	1.99	19297	28443	True	False
Number of rows in target table: 10 001 - 100 000								
Adve.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	47273	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

Table 5.3: List of regression datasets used in experiments with statistics.

5.3.2 Environment

All experiments executed on the blueprint instances discussed in Section 4.5 used a hyperparameter optimization pipeline with the parts introduced in Section 5.2. The pipeline consists of Ray[80], used for the distribution of the resources, parallelization, and general management of the model training, Optuna[79] provides an efficient algorithm for search in the hyperparameter space, and finally MLFlow aggregates the parameters and metrics of the training and validation actions.

5.3.3 Parametrization

There were 16 runs per blueprint instance per dataset executed inside the model's hyperparameter space provided by Optuna. Each of the runs was learning for at least 4000+ training steps on a standard 70:30 training validation split. All DL models used Adam[15] optimizer with a learning rate set as a hyperparameter in a logarithmic space on interval $\langle 0.00005, 0.002 \rangle$. HGSampling as described in Section 4.1.3 provided the data sampling where the batch size was parametrized by the dataset size, hyperparameter scale factor from exponential space on 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 will always be in interval $\langle 16, 16384 \rangle$.

Embeddings dimension D for the Embedder was also a parameter in the Optuna search space and can be defined as a choice from a set of $\{16, 32, 64\}$. The number of layers N inside the blueprint instances was set as a random integer from the set $\{1, 2, 3, 4, 5\}$.

The decision-making decoder MLP head was parametrized by the number of linear layers M that was 1, 2, or 3, where each hidden layer had 64 channels and by a flag whether to use a Batch Normalization after each of the hidden layers.

5.3.4 Overall Results

The overall results of the blueprint instances, as presented in the Section 4.5, compared against a selection of state-of-the-art methods are outlined in the tables 5.4 and 5.5. The blueprint in these tables stands for the best results recorded among all blueprint instances; hence, even the alternative versions from Section 5.3.5.

The naive baseline for the comparison is the MLP implemented as a blueprint instance with only a simple Embedder that uses *categorical* and *numerical* variables with simple embeddings for the *categorical* attributes and Linear Embedder for the *numerical* attributes, and importantly decoder MLP as a prediction head; hence, the MLP is parametrized in the same way as the blueprint decoder (Sec. 5.3.3). A big portion of the dataset is not available

to the tabular models; nevertheless, in cases where they are applicable, the MLP model performs reasonably well.

The RDN-boost (RDNb) belongs to the Statistical Relational Learning (SRL) (Sec. 3.2) category. The method utilizes relational dependencies. However, it needs to set up 'modes'[50], which were implemented rather simply, probably explaining the shaky performance. Importantly, the method does not scale very well, resulting in some missing values in the results. The issue with scalability is a common factor to the other methods used and also the reason for the missing values.

The 'getML system'[52] as a representative of propositionalization (Sec. 3.2.2) showed a strong performance with the FastProp[83] for the feature learning and XGBoost[27] as a predictor. Lastly, the neuro-symbolic method CILP++[84] also had good results, slightly outperforming the default getML method on the classification tasks but losing on the regression tasks. Notably, the CILP++ was emulated using the getML for propositionalization, and then MLP was applied to the transformed data.

Model accuracy in %

Dataset	**MLP**	**RDN-b**	**GetML**	**CILP++**	**Blueprint**
Carcinogenesis	N/A	59.18	47.96	69.39	**75.51**
CraftBeer	11.38	0.60	5.39	11.38	**58.08**
Dallas	49.23	49.23	86.15	83.08	**66.15**
financial	75.49	N/A	**97.06**	79.90	88.73
Mondial	N/A	39.34	N/A	N/A	**100.00**
MuskSmall	N/A	40.74	74.07	81.48	**100.00**
mutagenesis	96.43	83.93	80.36	92.86	**98.21**
Pima	N/A	68.70	N/A	N/A	**83.48**
PremierLeague	59.87	34.21	61.40	73.68	**99.53**
Toxicology	N/A	56.86	63.73	67.65	**73.53**
UW_std	92.79	91.57	69.88	66.27	**98.06**
WebKP	N/A	N/A	59.70	57.41	**60.12**
DCG	N/A	50.15	85.84	73.45	**100.00**
Same_gen	N/A	14.51	**100.00**	**100.00**	**100.00**
voc	78.88	50.02	N/A	N/A	**85.16**
PubMed	N/A	N/A	**85.51**	84.87	64.07
Accidents	77.40	N/A	N/A	N/A	**93.20**
imdb_ijs	64.23	37.19	**94.39**	94.36	93.29
tpcd	20.90	N/A	N/A	N/A	**73.35**

Table 5.4: Total results of all different model types on the classification datasets.

In general, the blueprint instances outperformed the selection of the methods with a small number of exceptions where the propositionalization shined. The performance of each type of the blueprint instances is outlined in the table 5.6.

The best results were displayed by the proposed Database Transformer model, demonstrating the strength of the Transformer architecture. SAGE GNN, TabNet and SAINT presented a good performance as well. TabTransfomer and Trompt had weaker results but, with slightly adjusted blueprint components, could present a reasonable choice as well.

To further improve the understanding of the importance of various possible alternative adjustments in the blueprint instances, the Database Transformer was trained in various settings. The next section will test the effect of some of the possible variations and prove the impact of the changes by displaying a great boost in performance.

Model NRMSE

Dataset	**MLP**	**GetML**	**CILP++**	**Blueprint**
classicmodels	0.58	0.65	1.19	**0.16**
GOSales	N/A	N/A	N/A	**0.17**
northwind	1.10	1.16	1.36	**0.10**
Triazine	N/A	0.20	0.18	**0.12**
Basketball_men	0.20	0.23	0.25	**0.17**
restbase	0.19	0.19	0.20	**0.07**
AdventureWorks	0.03	0.05	3.29	**0.01**
FNHK	0.83	0.65	0.69	**0.06**
sakila	0.54	N/A	N/A	**0.36**
stats	0.95	2.59	6.47	**0.14**
Grants	2.432	N/A	N/A	**2.429**
ConsumerExpend.	6.38	**6.26**	7.37	6.33
employee	0.27	N/A	N/A	**0.25**
SalesDB	N/A	N/A	N/A	**0.13**
Seznam	5.34	N/A	6.13	**3.41**

Table 5.5: Total results of all different model types on the regression datasets. A lower number is better.

Dataset	DB Trans.	SAGE GNN	SAINT	Tab Net	Tab Trans.	Trompt
Classification						
Model accuracy in %						
Carcinoge.	71.43	69.39	72.45	**73.47**	70.41	69.39
CraftBeer	12.57	**14.97**	13.17	**14.97**	13.77	13.17
Dallas	55.38	55.38	56.92	**66.15**	56.92	58.46
financial	74.02	78.39	74.06	**79.41**	75.98	78.92
Mondial	98.94	93.44	96.72	96.72	94.07	**98.95**
MuskSmall	96.30	96.30	88.89	96.30	88.89	**100**
mutagen.	96.43	**98.21**	94.64	98.21	96.43	96.43
Pima	80.87	80.43	81.30	**83.48**	80.00	80.87
Prem.Leag.	74.79	**82.49**	59.18	71.69	66.25	59.76
Toxicology	**73.53**	70.59	71.57	**73.53**	71.57	71.57
UW_std	93.51	**98.06**	93.39	86.73	86.90	85.98
WebKP	56.40	56.16	**60.12**	53.55	52.13	54.30
DCG	89.09	62.24	64.60	94.10	69.91	**100**
Same_gen	92.97	89.74	**93.38**	89.57	90.57	88.46
voc	**79.46**	79.13	74.58	67.90	76.34	68.59
PubMed	54.93	55.22	61.56	52.48	**64.07**	61.62
Accidents	77.56	**78.70**	77.75	77.43	78.16	78.22
imdb_ijs	64.12	63.73	63.51	64.04	**64.16**	63.27
tpcd	21.26	**22.60**	21.08	21.19	21.00	21.40
Regression						
Model NRMSE						
Biodegrad.	**0.15**	0.18	0.17	0.17	0.18	0.16
classicmod.	0.50	0.49	0.46	**0.40**	0.46	1.09
GOSales	0.42	0.53	**0.40**	0.52	0.79	0.75
northwind	**0.48**	0.74	0.80	0.88	0.86	0.97
Triazine	0.14	0.16	**0.12**	0.17	0.18	0.14
Basketball	0.23	0.23	0.28	**0.21**	0.25	0.26
restbase	0.18	0.19	**0.17**	0.18	0.18	0.18
Adve.Works	**0.01**	0.06	2.17	2.99	0.34	2.38
FNHK	0.80	0.80	**0.73**	0.80	1.00	0.75
sakila	0.52	0.49	**0.47**	0.55	0.56	0.52
stats	**0.14**	1.65	0.29	2.95	3.00	2.98
Grants	3.73	3.75	**2.43**	3.07	2.69	3.29
Consu.Ex.	6.36	6.36	**6.34**	6.64	6.75	6.76
employee	**0.26**	0.26	0.50	0.27	0.26	0.71
SalesDB	**0.42**	0.51	0.55	0.51	0.44	0.55
Seznam	3.66	3.94	4.68	4.32	**3.41**	4.04
Avg. Rank	**2.63**	**3.37**	**3.26**	**3.37**	**3.97**	**3.86**

Table 5.6: Total results of various blueprint instances that were considered in section 4.5. DB Transformer here is the baseline version.

5.3.5 Database Transformer Versions

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 5.7: Overview of the baseline DB Transformer to the DB Transformer utilizing the text embeddings. Models are compared on the datasets containing textual non-key attributes.

With LLMs being ever more powerful, the plain text data can be utilized with great precision if the information value is present within. This is not an exception in the settings of relational databases. Table 5.7 summarizes the performance of the Database Transformer in the baseline settings to the one with leveraging the text embeddings. The text embeddings help the Database

Transformer in a significant manner and outperform the baseline model over the board with a couple of small exceptions.

The work done in the RelBench[60] project heavily focuses on the time attributes in the relational databases. To evaluate the importance of the time features in the data, the table 5.8 shows a comparison of the DB Transformer model with a Timestamp Embedder utilizing the time attributes to the baseline version. The Timestamp Embedder strongly improves the performance on almost all relevant datasets, hence validating the importance of 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 5.8: Summary of the baseline DB Transformer to the DB Transformer utilizing the Timestamp Embedder. Models are compared using datasets containing time attributes.

Employing both text and time attributes showed significant improvements in performance. To possibly advance the accuracy of the blueprint

instance even further, there are two other options that were considered. First, enlarging the number of neighboring nodes in the sampled graph provided by HGSampling as it is possible that the graph was undersampled with too few connections that do not fully describe the nature of the references engaged. Secondly, by adding *positional encoding*. It can be argued that positional encodings are only important when a sequence of features of the same type is involved[24]. As that is not the case for the tabular nature of the relational database data, it should not be used.

Table 5.9 shows the results of using the DB Transformer baseline trained with a larger graph neighborhood and the DB Transformer utilizing positional encoding to the baseline model trained with a standard neighborhood. Results prove that some of the datasets were undersampled, but the majority of the datasets saw no improvement or even a decrease in precision, hence prompting the possibility of another hyperparameter to optimize. Usage of positional encodings showed minor improvements on smaller datasets, although, overall, it seemed rather insignificant for the model's performance, thus at least partially proving the hypothesis that *positional encoding* is not valuable in this context.

Classification

Model accuracy in %

Dataset	Baseline	More Neigh.	Impr. Neigh.	Positional	Impr. Pos.
Carcinoge.	71.43	71.43	0.00	75.51	**4.08**
CraftBeer	12.57	12.57	0.00	14.97	**2.40**
Dallas	55.38	55.38	0.00	58.46	**3.08**
financial	74.02	75.49	**1.47**	77.14	**3.12**
Mondial	98.94	78.69	-20.25	98.97	0.03
MuskSmall	96.30	96.30	0.00	96.30	0.00
mutagen.	96.43	96.43	0.00	96.43	0.00
Pima	80.87	83.04	**2.17**	80.00	-0.87
Prem.Leag.	74.79	61.32	-13.47	67.68	-7.11
Toxicology	73.53	68.63	-4.90	71.57	-1.96
UW_std	93.51	97.37	**3.86**	94.02	**0.51**
WebKP	56.40	54.58	-1.82	55.59	-0.81
DCG	89.09	98.82	**9.73**	92.92	**3.83**
Same_gen	92.97	100.00	**7.03**	90.33	-2.64
voc	79.46	80.03	**0.57**	79.09	-0.37
PubMed	54.93	56.42	**1.49**	53.61	-1.32
Accidents	77.56	93.20	**15.64**	77.55	-0.01
imdb_ijs	64.12	63.60	-0.52	63.03	-1.09
tpcd	21.26	21.56	**0.30**	21.57	**0.31**

Regression

Model NRMSE

Dataset	Baseline	More Neigh.	Decr. Neigh.	Positional	Decr. Pos.
Biodegrad.	0.15	0.17	0.01	0.15	0.00
classicmod.	0.50	0.54	0.04	0.53	0.03
GOSales	0.42	0.73	0.31	0.38	-0.04
northwind	0.48	1.10	0.61	0.57	0.09
Triazine	0.14	0.17	0.03	0.15	0.01
Basketball	0.23	0.34	0.11	0.26	0.03
restbase	0.18	0.16	-0.02	0.18	0.00
Adve.Works	0.01	0.14	0.13	0.49	0.48
FNHK	0.80	0.77	-0.02	0.77	-0.03
sakila	0.52	0.46	-0.06	0.52	0.01
stats	0.14	1.96	1.82	0.35	0.21
Grants	3.73	6.11	2.38	5.09	1.36
Consu.Ex.	6.36	6.65	0.29	6.33	-0.02
employee	0.26	0.27	0.00	0.26	0.00
SalesDB	0.42	1.00	0.59	0.44	0.03
Seznam	3.66	5.31	1.65	4.17	0.51

Table 5.9: Summary of the baseline DB Transformer to the DB Transformer trained with a greater amount of neighboring nodes and to the DB Transformer model with a *positional encoding* module in the Post-Embedder.

Chapter 6

Conclusion

The overarching objective of this thesis was to advance the understanding and application of deep learning technologies on relational databases by leveraging their inherent *heterogeneous tabular graph* structure. Through a detailed exploration and integration of principles from tabular models, graph neural networks, and transformer architecture, this work has endeavored to bridge the gap between traditional deep learning approaches and the rich interconnected data encapsulated within relational databases.

The successful contextualization of relational databases as *heterogeneous tabular graphs* was an important achievement. Doing so laid a solid foundation for applying deep learning techniques directly to the relational data. This paradigm shift is crucial for preserving the structural and relational integrity of the data, which is often compromised in traditional pre-processing methods.

Having described the relational databases in terms of graph structures, the *blueprint* model has shown the utilization of this definition to provide a general structure of the deep learning model for relational databases. Its modular yet structurally predefined design creates a vast neural architecture space. Notably, this allows the integration of various tabular models with ranging inner representation structures.

To highlight the blueprint structure, a Database Transformer model was proposed, combining features of transformers with heterogeneous GNNs. Database Transformer employs both *self-attention* and *cross-attention* to lever-

age the sequence-like structure of encoded tuples and their interconnections. Further research of the architecture space was established by extensions of existing tabular models, often transformer-based, to the settings of relational databases.

The conducted experiments tested the proposed blueprint models as well as the deep learning pipeline. The pipeline allows for end-to-end integration, consisting of a database connection layer, the schema auto-detection, data loading, graph creation, data sampling, and the blueprint itself. This allows for a streamlined deep-learning process, allowing one to focus on the model design rather than the nuances of data processing. Notably, the pipeline is also directly integrated with the CTU Prague Relational Learning Repository. This allows for the broad variance of datasets as well as a direct comparison of future models with the results of this thesis.

The experiments also showcased the performance of the blueprint instances in comparison to state-of-the-art methods from related fields such as propositionalization, statistical relational learning, and neuro-symbolic integration. The experimental outcomes depict promising results as the blueprint instances generally outperform the existing models. Especially the proposed Database Transformer displayed a superior performance. The alternative versions of Database Transformer highlighted the possibilities of utilizing cyclic encoding for time attributes and language models for text attributes, where both cases led to a dramatic boost in accuracy.

The integration of deep learning with relational databases opens new horizons for machine learning. With continued research and development, the potential of this approach can be fully realized, leading to significant advancements in the field of artificial intelligence.

6.1 Future Work

Although the concluded successes of the thesis are numerous, there are still plenty of directions to evaluate in the future. Ranging from further research in the blueprint architecture space to the real-world application of the laid foundations.

6.1.1 Blueprint instances

The vast architectural space of the blueprint allows for the creation of countless viable models. The research done in this thesis experimented only with a small subset of the options that can be utilized. To fully realize the potential of the proposed methodology, further experiments need to be conducted. Namely, the employment of the tabular models leaves many possibilities on the table. For instance the integration of the tabular model as an initial encoder in the Post-Embedder module with the Database Transformer architecture.

Also, the Database Transformer, as well as the other blueprint instances, were tested with a hyperparameter optimization routine. This allowed for finding the best hyperparameters for a given model. The analysis of the found hyperparameters could uncover the possibility to fix some of the parameters hence creating standardized models that perform generally well on any dataset.

6.1.2 Enriching the RelBench Package

As mentioned in Section 5.1.1, the contributions of RelBench [60] towards establishing Relational Deep Learning as a distinct subfield within deep learning are significant. However, the current version of RelBench is limited in terms of dataset variety. A viable solution to enhance its utility is by integrating the CTU Relational Repository datasets into the package. Additionally, RelBench presently lacks functionalities for direct interaction with relational database data and for conducting comprehensive analyses of data schemas. Expanding RelBench to include these capabilities would significantly benefit ongoing research.

6.1.3 Pre-training

Pre-training is a technique in deep learning (and other machine learning fields), where a model is initially trained on a broader task to learn general patterns before being fine-tuned on a more specific goal. This approach leverages the extensive knowledge gained during the initial training phase to improve performance on specific tasks.

This method is often used on the tabular models via the self-supervised training approach, where some table values are corrupted or masked, and the goal of the model is to detect such changes or predict the missing values. A similar scheme could be applied to the setting of relational database data to the extent that the masking or cell corruption would be applied to all tables in the database.

Furthermore, pre-training on multiple databases with the use of a large model could be possible. This would allow for the creation of a base pre-trained relational database model in some sense similar to the large language models. Such a model could be utilized even on new databases with a limited number of records without the need for fine-tuning.

6.1.4 Temporal Data

Records stored in relational databases are usually gradually added over time. This suggests an option to train models on dataset split containing only records that were created before a predefined timestamp and validate on the rest. The experiments with the Database Transformer also proved the importance of the time attributes; hence, the temporal training splits might be an interesting direction for further research.

6.1.5 Attention encoding of SQL statements

SQL `SELECT` statement is used to query and retrieve specific data from a database. While the attention mechanism and SQL `SELECT` statements operate in different domains, they share a conceptual similarity in their approach to focusing on and retrieving relevant information from a larger set of data. However, their methodologies differ significantly. The attention mechanism excels in processing sequential and interconnected data. In contrast, SQL `SELECT` statements offer a structured, predefined approach to data retrieval in the context of relational databases.

This prompts a question of what will happen inside the attention weights when a model like the Database Transformer is trained on a task where the target values are generated through a SQL `SELECT` statement. Importantly,

the reversed goal can be considered as to map the weights to a `SELECT` statement describing the target values. Analysis of attention weights on such tasks might prove to be valuable for encoding pre-defined logic into the model[85]. The ability to achieve such encoding might also be practical for the creation of a general pre-trained database model.

6.1.6 Real-world application

Relational databases store the majority of the world's data, yet very few of them are publicly available. This strikes as a limitation for testing the accuracy of the models on the actual production grade data; thus, the real trial for the viability of proposed approaches is the application in the wild.

Bibliography

[1] E. F. Codd, "A relational model of data for large shared data banks," *Commun. ACM*, vol. 13, p. 377–387, 6 1970.

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

[3] T. Haerder and A. Reuter, "Principles of transaction-oriented database recovery," *ACM Comput. Surv.*, vol. 15, p. 287–317, 12 1983.

[4] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, 05 2015.

[5] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 10 2017.

[6] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015.

[7] A. Ivakhnenko, "Heuristic self-organization in problems of engineering cybernetics," *Automatica*, vol. 6, no. 2, pp. 207–219, 1970.

[8] Y. LeCun, "1.1 deep learning hardware: Past, present, and future," in *2019 IEEE International Solid-State Circuits Conference - (ISSCC)*,

pp. 12–19, 2019.

[9] H. Chen, O. Engkvist, Y. Wang, M. Olivecrona, and T. Blaschke, "The rise of deep learning in drug discovery," *Drug Discovery Today*, vol. 23, no. 6, pp. 1241–1250, 2018.

[10] O. B. Sezer, M. U. Gudelek, and A. M. Ozbayoglu, "Financial time series forecasting with deep learning : A systematic literature review: 2005–2019," *Applied Soft Computing*, vol. 90, p. 106181, 2020.

[11] A. Gupta, A. Anpalagan, L. Guan, and A. S. Khwaja, "Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues," *Array*, vol. 10, p. 100057, 2021.

[12] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[13] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," 2016.

[14] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, no. 4-5, pp. 185–196, 1993.

[15] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[16] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 2015.

[17] H. Salehinejad, S. Sankar, J. Barfett, E. Colak, and S. Valaee, "Recent advances in recurrent neural networks," 2018.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017.

[19] L. Floridi and M. Chiriatti, "Gpt-3: Its nature, scope, limits, and consequences," *Minds and Machines*, vol. 30, pp. 681–694, 12 2020.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training

of deep bidirectional transformers for language understanding," 2019.

[21] 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, 2009.

[22] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[23] G. Karystinos and D. Pados, "On overfitting, generalization, and randomly expanded training sets," *IEEE Transactions on Neural Networks*, vol. 11, no. 5, pp. 1050–1057, 2000.

[24] 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.

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

[26] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *CoRR*, vol. abs/2110.01889, 2021.

[27] 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*, KDD '16, ACM, Aug. 2016.

[28] N. Lavrač, V. Podpečan, and M. Robnik-Šikonja, *Machine Learning Background*, pp. 17–53. Cham: Springer International Publishing, 2021.

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

[30] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[31] D. Zhu, J. Chen, X. Shen, X. Li, and M. Elhoseiny, "Minigpt-4: En-

hancing vision-language understanding with advanced large language models," 2023.

[32] A. F. Karr, A. P. Sanil, and D. L. Banks, "Data quality: A statistical perspective," *Statistical Methodology*, vol. 3, no. 2, pp. 137–173, 2006.

[33] Y. Gorishniy, I. Rubachev, and A. Babenko, "On embeddings for numerical features in tabular deep learning," 2023.

[34] I. Shavitt and E. Segal, "Regularization learning networks: deep learning for tabular datasets," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[35] J. Yoon, Y. Zhang, J. Jordon, and M. Van der Schaar, "Vime: Extending the success of self-and semi-supervised learning to tabular domain," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11033–11043, 2020.

[36] B. Sun, L. Yang, W. Zhang, M. Lin, P. Dong, C. Young, and J. Dong, "Supertml: Two-dimensional word embedding for the precognition on structured tabular data," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 0–0, 2019.

[37] Y. Zhu, T. Brettin, F. Xia, A. Partin, M. Shukla, H. Yoo, Y. A. Evrard, J. H. Doroshow, and R. L. Stevens, "Converting tabular data into images for deep learning with convolutional neural networks," *Scientific reports*, vol. 11, no. 1, p. 11325, 2021.

[38] 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.

[39] J. Berant, D. Deutch, A. Globerson, T. Milo, and T. Wolfson, "Explaining queries over web tables to non-experts," 2018.

[40] P. Yin, G. Neubig, W. Yih, and S. Riedel, "Tabert: Pretraining for joint understanding of textual and tabular data," *CoRR*, vol. abs/2005.08314, 2020.

[41] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisensch-

los, "TAPAS: weakly supervised table parsing via pre-training," *CoRR*, vol. abs/2004.02349, 2020.

[42] H. Iida, D. Thai, V. Manjunatha, and M. Iyyer, "TABBIE: pretrained representations of tabular data," *CoRR*, vol. abs/2105.02584, 2021.

[43] Z. Wang, H. Dong, R. Jia, J. Li, Z. Fu, S. Han, and D. Zhang, "Structure-aware pre-training for table understanding with tree-based transformers," *CoRR*, vol. abs/2010.12537, 2020.

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

[45] J. Chen, J. Yan, D. Z. Chen, and J. Wu, "Excelformer: A neural network surpassing gbdts on tabular data," 2023.

[46] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: unbiased boosting with categorical features," 2019.

[47] L. Getoor and B. Taskar, *Introduction to Statistical Relational Learning*. The MIT Press, 08 2007.

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

[49] S. Muggleton and L. De Raedt, "Inductive logic programming: Theory and methods," *The Journal of Logic Programming*, vol. 19, pp. 629–679, 1994.

[50] 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.

[51] S. Kramer, N. Lavrač, and P. Flach, *Propositionalization Approaches to Relational Data Mining*, pp. 262–291. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001.

[52] The SQLNet Company GmbH, "getml."

[53] G. Šourek, F. Železný, and O. Kuželka, "Beyond graph neural networks with lifted relational neural networks," *Machine Learning*, vol. 110, p. 1695–1738, June 2021.

[54] M. V. França, G. Zaverucha, and A. S. D'avila Garcez, "Fast relational learning using bottom clause propositionalization with artificial neural networks," *Mach. Learn.*, vol. 94, p. 81–104, jan 2014.

[55] 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, 2009.

[56] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," 2017.

[57] Y. Feng, H. You, Z. Zhang, R. Ji, and Y. Gao, "Hypergraph neural networks," 2019.

[58] M. Cvitkovic, "Supervised learning on relational databases with graph neural networks," 2020.

[59] 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*, WWW '21, (New York, NY, USA), p. 1540–1551, Association for Computing Machinery, 2021.

[60] M. Fey, W. Hu, K. Huang, J. E. Lenssen, R. Ranjan, J. Robinson, R. Ying, J. You, and J. Leskovec, "Relational deep learning: Graph representation learning on relational databases," 2023.

[61] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of json schema," in *Proceedings of the 25th International Conference on World Wide Web*, pp. 263–273, International World Wide Web Conferences Steering Committee, 2016.

[62] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," 2019.

[63] 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.

[64] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," in *NIPS-W*, 2017.

[65] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," 2019.

[66] S. Lin, "Computer solutions of the traveling salesman problem," *Bell System Technical Journal*, vol. 44, no. 10, pp. 2245–2269, 1965.

[67] A. Bundy and L. Wallen, *Breadth-First Search*, pp. 13–13. Berlin, Heidelberg: Springer Berlin Heidelberg, 1984.

[68] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 2018.

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

[70] 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.

[71] 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*, KDD '19, (New York, NY, USA), p. 793–803, Association for Computing Machinery, 2019.

[72] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016.

[73] D. Aha, "UCI Machine Learning Repository." UCI Machine Learning Repository, 1987.

[74] A. Johnson, L. Bulgarelli, T. Pollard, S. Horng, L. Celi, and R. Mark, "Mimic-iv (version 1.0)," 2020.

[75] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network

dataset collection." `http://snap.stanford.edu/data`, June 2014.

[76] 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.

[77] J. Motl and O. Schulte, "The CTU prague relational learning repository," *CoRR*, vol. abs/1511.03086, 2015.

[78] M. Feurer and F. Hutter, *Hyperparameter Optimization*, pp. 3–33. Cham: Springer International Publishing, 2019.

[79] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," 2019.

[80] 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.

[81] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[82] P.-T. de Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein, "A tutorial on the cross-entropy method," *Annals of Operations Research*, vol. 134, pp. 19–67, Feb 2005.

[83] N. Sören, "Introducing FastProp: the fastest approach to propositionalization," 6 2021.

[84] 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.

[85] 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.

Appendix A

Glossary

BFS Breadth-First Search. 28

CNN Convolutional Neural Network. 7, 17

CSV Comma-separated values. 42

DL Deep Learning. 7, 33, 50

DNN Deep Neural Network. 7, 17, 26

GNN Graph Neural Network. 12, 13, 15, 22–24, 29, 34, 42, 59

GPU Graphics processing unit. 26

i.i.d. independent and identically distributed. 27

JSON JavaScript Object Notation. 26

LLM Large Language Model. 8, 54

ML Machine Learning. 19

MLP Multilayer perceptron. 48, 50

MSE Mean squared error. 48

NN Neural Network. 7

PyFrame PyTorch Frame. 27

PyG PyTorch Geometric. 27

RDB Relational Database. 3

RDBMS Relational Database Management System. 2, 6, 23, 24

SQL Structured Query Language. 6, 25, 28

SRL Statistical Relational Learning. 15

TNN Tabular Neural Network. 16

Appendix B

Supplementary material

- All code used for this thesis can be found on GitHub repository `https://github.com/jakubpeleska/deep-db-learning`.
- All datasets used for the experiments with some additional information can be found at `https://relational.fel.cvut.cz`.

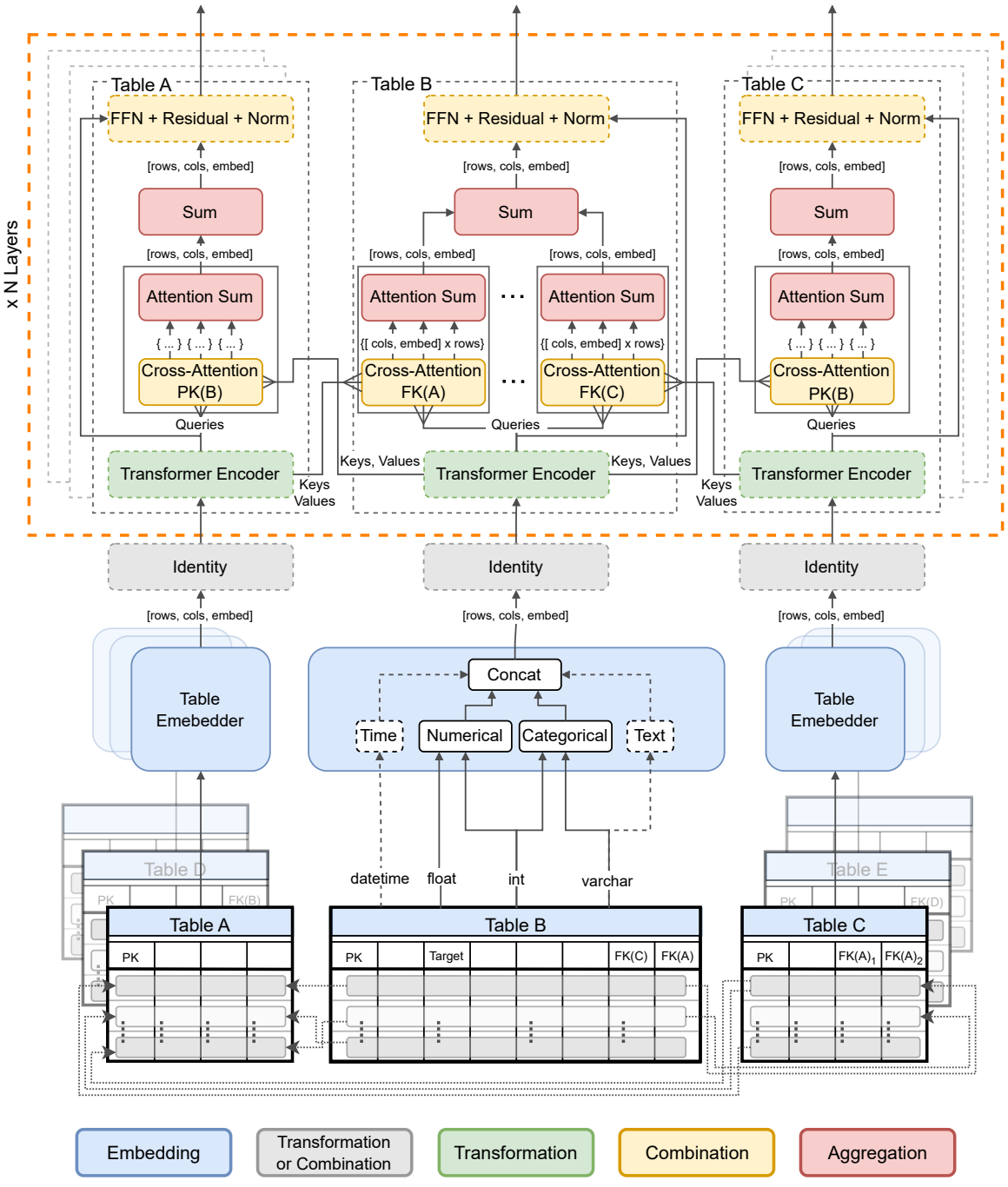


Figure B.1: An alternative depiction of Database Transformer (Sec. 4.4) focusing on the categories of functions used inside the blueprint modules (Sec. 4.2).