Graph Machine Learning Meets Multi-Table Relational Data

Quan Gan
Amazon
Shanghai, China
quagan@amazon.com

Minjie Wang
Amazon
Shanghai, China
minjiw@amazon.com

David Wipf
Amazon
Shanghai, China
daviwipf@amazon.com

Christos Faloutsos
CMU and Amazon
Pittsburgh, PA USA
faloutso@amazon.com

ABSTRACT

While graph machine learning, and notably graph neural networks (GNNs), have gained immense traction in recent years, application is predicated on access to a *known input graph* upon which predictive models can be trained. And indeed, within the most widely-studied public evaluation benchmarks such graphs are provided, with performance comparisons conditioned on curated data explicitly adhering to this graph. However, in real-world industrial applications, the situation is often quite different. Instead of a known graph, data are originally collected and stored across multiple tables in a repository, at times with ambiguous or incomplete relational structure. As such, to leverage the latest GNN architectures it is then up to a skilled data scientist to first manually construct a graph using intuition and domain knowledge, a laborious process that may discourage adoption in the first place. To narrow this gap and broaden the applicability of graph ML, we survey existing tools and strategies that can be combined to address the more fundamental problem of predictive tabular modeling over data native to multiple tables, with no explicit relational structure assumed a priori. This involves tracing a comprehensive path through related table join discovery and fuzzy table joining, column alignment, automated relational database (RDB) construction, extracting graphs from RDBs, graph sampling, and finally, graph-centric trainable predictive architectures. Although efforts to build deployable systems that integrate all of these components while minimizing manual effort remain in their infancy, this survey will nonetheless reduce barriers to entry and help steer the graph ML community towards promising research directions and wider real-world impact.

CCS CONCEPTS

• **Computing methodologies** → **Supervised learning**.

KEYWORDS

graph machine learning, graph neural networks, relational databases, tabular modeling, data augmentation

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08.
https://doi.org/10.1145/3637528.3671471

ACM Reference Format:
Quan Gan, Minjie Wang, David Wipf, and Christos Faloutsos. 2024. Graph Machine Learning Meets Multi-Table Relational Data. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3637528.3671471

1 INTRODUCTION

Graph machine learning, and in particular graph neural networks (GNNs) [7, 40, 45, 50, 56, 59, 83, 93], have made remarkable inroads across predictive modeling tasks involving large-scale relational data drawn from recommender systems [28, 35, 99, 103], social networks [8, 66, 90], and fraud or anomaly detection applications [22, 58, 73, 91, 109] among many others. However, these advancements nonetheless belie a significant shortcoming: Generally speaking, to actually apply powerful GNN or comparable architectures, *an explicit input graph must first be provided*, after which the model can be trained to minimize an objective of interest, such as node classification [74] or link prediction [111]. While in the confines of academic research and public benchmarks such graphs are regularly provided (e.g., [24, 49, 51, 57, 75]), in real-world industrial applications data is frequently formatted as a collection of tables with unknown or incomplete relational structure; see Figure 1 for a real-world example. And even when some degree of relational structure is present, such as within a relational database (RDB) [36], there is still no unique way of mapping to an input graph, and so lingering uncertainty remains.

The deleterious consequences of this mismatch are (at least) three-fold. First, GNN successes thus far are largely predicated on an experienced data scientist constructing a graph using domain-specific ingenuity and laborious, manual inspection of table contents. As this process can take months of iterations for large-scale data repositories, along the way non-trivial resources will invariably be consumed. Secondly though, and perhaps more difficult to quantify, are the number of situations whereby insufficiently experienced personnel make an initial attempt to apply GNNs, but fail for one reason or another, often at the graph construction stage. Indeed our industry colleagues have observed variants of this scenario unfolding on a regular basis (which is the original motivation for this survey). And finally, given the growing complexity and scale of relational data, projecting forward we may encounter situations where even the most experienced domain specialists struggle to make headway without explicitly-granted graph structure. As such, a reasonable case can be made that, were we to zoom out and

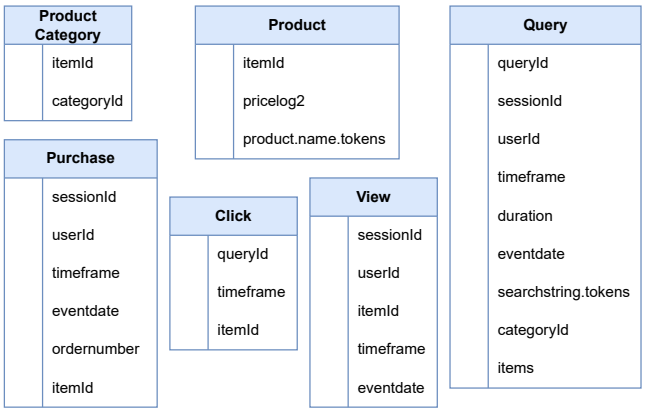

Figure 1: Table and column names of the Diginetica dataset [17]. No schema information is provided. Although some columns (e.g., `itemId`, `queryId`) are likely foreign/primary keys as discussed in Section 3.1, others are less apparent (e.g., `ordernumber`).

directly confront raw multi-table data head on, more durable gains and broader applicability of graph ML could be established.

Along these lines, it is worthwhile to note that there already exists a wide body of research for processing and automatically linking data spanning multiple tables [19, 21, 27, 46, 62, 69, 89, 113, 118]. Instead, what is largely missing are the intermediate steps needed to effectively integrate these efforts from the tabular side, with the latest GNN architectures and modeling techniques from the graph ML side. It is with these considerations in mind that our survey herein addresses the fundamental problem of predictive tabular modeling of unknown quantities within data native to multiple tables, with no explicit relational structure assumed a priori. Starting from the vantage point of raw tables, we first define a generic family of predictive tasks and practical scenarios that motivate them in Section 2. Then, in Section 3 we piece together the construction of an RDB using automated tools for join discovery and alignment of related columns across tables. The motivation here is that RDBs serve as a natural intermediate abstraction between completely unstructured tables, and fully-specified graphs conducive to graph ML. With respect to the latter, we introduce the RDB-to-graph conversion process in Section 4. And to round things out, trainable graph-based architectures are considered in Section 5, followed by a discussion of future trends, including large language models, in Section 6.

We remark that the journey from raw collections of tables within a data repository, to powerful graph-based predictive models over targets of interest, involves a number of subtle, interrelated steps. In this regard, there exists high-quality surveys (and related comparative papers) covering some of these steps in isolation, such as single-table tabular prediction [2, 5, 29, 42], related table discovery and fuzzy table joining [26, 61], GNN models (but conditioned on a given graph) [100, 117], applying GNNs to established RDBs [31] or single tables [68]. But to our knowledge there is not yet an integrated treatment that covers the entire spectrum as is the primary focus herein.

2 PREDICTIVE MODELING ON MULTI-TABLE RELATIONAL DATA

In this section we first introduce the high-level form of predictive modeling over multi-table relational data that we wish to survey. By design, the setup we adopt is sufficiently general to subsume the majority of existing work in the literature as notable instances. Next, after motivating practical multi-table scenarios, we describe training and inference objectives, followed by an overview of modeling components utilized to operationalize these objectives. Later sections will then fill out details of these individual components.

2.1 High-Level Problem Definition

Our starting assumption is that any collection of tables is time-varying, with rows and/or columns regularly being *added* or *removed* (or possibly edited). To reflect this inherent dynamism, we denote a set of K tables at time/state $s \in \mathcal{S}$ as

$$\mathcal{D}(s) \;:=\; \left\{\boldsymbol{T}^k(s)\right\}_{k=1}^{K}, \tag{1}$$

where $\mathcal{S}$ is a set of candidate states and $\boldsymbol{T}^k(s) \in \mathbb{R}^{n^k(s)\times d^k(s)}$ refers to the k-th constituent table with $n^k(s)$ rows and $d^k(s)$ columns (note that we will sometimes omit dependency on s when the context and meaning are clear). We adopt $\boldsymbol{T}^k_{i:}$ and $\boldsymbol{T}^k_{:j}$ to reference the i-the row and j-th column of $\boldsymbol{T}^k$ respectively. While not strictly necessary, it is generally assumed that each column within any given table designates attribute values of a consistent type, e.g., numerical, categorical, text, while each row represents an instance [2, 5]. We then have the following core objective:[1]

> **Problem definition**: *Using all relevant information available in $\mathcal{D}(s)$, predict unknown quantities of interest $\boldsymbol{T}^k_{ij}(s)$ as uniquely specified by the tuple $\{s, k, i, j\}$, where s determines the state, k the table, and $\{i, j\}$ the table cell we wish to estimate.*

We emphasize that i may depend on s if rows are being added to table k, so this problem definition covers a broad spectrum of predictive tasks. Columns may be added as well; however, for a given predictive model we will assume both j and k are fixed.

To actualize such predictions, we ideally would like to train a model to approximate the distribution $p\left(\boldsymbol{T}^k_{ij}(s) \mid \mathcal{D}(s) \setminus \boldsymbol{T}^k_{ij}(s)\right)$, which is merely the distribution of the unknown target conditioned on all other information available spanning $\mathcal{D}(s)$. Indeed canonical tabular prediction tasks, involving a *single table* $\boldsymbol{T}^1$ stipulated to have iid rows and an unknown target column j, naturally fall under this paradigm with $K = 1$ and the simplified posterior

$$p\left(\boldsymbol{T}^1_{ij}(s) \mid \mathcal{D}(s) \setminus \boldsymbol{T}^1_{ij}(s)\right) = p\left(\boldsymbol{T}^1_{ij}(s) \mid \boldsymbol{T}^1_{i:}(s) \setminus \boldsymbol{T}^1_{ij}(s)\right), \tag{2}$$

for any row i available at state s. We remark that the r.h.s. of (2) simply denotes the distribution of $\boldsymbol{T}^1_{ij}(s)$ conditioned on all other elements in the i-th row of table $\boldsymbol{T}^1(s)$.

[1] For reference, Figure 2 (far left-hand side) shows a simple example with three tables.

2.2 Practical Multi-Table Data Scenarios

Multi-table data is ubiquitous across numerous application domains with immense practical significance. We now highlight representative scenarios that naturally give rise to such data.

Data Augmentation from Unstructured Repositories. Tabular prediction tasks in the past have most commonly been restricted to single-table data as in (2), with considerable effort devoted towards producing better approximations of $p\left(\boldsymbol{T}^1_{ij}(s) \mid \boldsymbol{T}^1_{i:}(s) \setminus \boldsymbol{T}^1_{ij}(s)\right)$. However, even if oracle knowledge of the optimal distribution were somehow known, a simple bias-variance decomposition reveals that only the estimation *bias* can generally be driven to zero. In contrast, the *variance* can remain large leading to poor performance. And the only way to reduce such troublesome variance is to condition on additional information such as may be available in auxiliary tables within a larger data repository, i.e., data augmentation [13, 71]. In fact, for any table k within some $\mathcal{D}(s)$ with $K > 1$, it always holds (at least in expectation [37]) that

$$\text{var}\left[\boldsymbol{T}^k_{ij}(s) \mid \boldsymbol{T}^k_{i:}(s) \setminus \boldsymbol{T}^k_{ij}(s)\right] \;\geq\; \text{var}\left[\boldsymbol{T}^k_{ij}(s) \mid \mathcal{D}(s) \setminus \boldsymbol{T}^1_{ij}(s)\right]. \quad (3)$$

Hence reducing variance (and ultimately prediction errors), can be accomplished by conditioning on data from relevant auxiliary tables included within $\mathcal{D}(s)$.

Predictive Analytics over Relational Databases. RDBs constitute a widely-occurring special case of $\mathcal{D}(s)$ whereby certain columns of the constituent tables designate known inter-table links, or so-called primary key/foreign key pairs as will be further detailed in Section 3. Relevant predictive tasks include missing value imputation, recommendation, and forecasting future RDB quantities of interest. For example, on e-commerce or related online platforms representative prediction targets might include future product purchases [17, 78], customer retention [18], click-through rates [17, 76, 120], user churn [78, 86], or charge/pre-payment attributes [77]. Combined with the growing enterprise reliance on RDBs (e.g., just the market for RDB management systems is expected to exceed $133 billion USD by 2028 [82]), building robust models for addressing these tasks represents a key emerging frontier facing the machine learning and data mining communities.

2.3 Training and Inference Specifications

Training. Because the true target distribution $p\left(\boldsymbol{T}^k_{ij}(s) \mid \mathcal{D}(s) \setminus \boldsymbol{T}^k_{ij}(s)\right)$ for any non-trivial predictive task is generally unknown, a trainable approximation is needed. For a given predictive task defined by table k and target column j, this can be accomplished by minimizing a negative log-likelihood objective as in

$$\min_{\theta} \sum_{s \in \mathcal{S}_{tr}} \sum_{i \in \psi_{tr}(s)} -\log \hat{p}\left(\boldsymbol{T}^k_{ij}(s) \mid f\left[\mathcal{D}(s) \setminus \boldsymbol{T}^k_{ij}(s); \theta\right]\right). \quad (4)$$

In this expression, θ represents parameters of a model $f(\,\cdot\,;\theta)$ whose output specifies the approximate predictive distribution $\hat{p}$. For example, $\hat{p}$ can be chosen as a Gaussian with conditional mean given by f for continuous-value predictions, or a Bernoulli distribution with logits f for binary classification. Meanwhile, each training instance is indexed by an element of the set $\{(i,s) : i \in \psi_{tr}(s), s \in \mathcal{S}_{tr}\}$. Here $\mathcal{S}_{tr}$ represents a set of states containing rows i within the s-dependent set $\psi_{tr}(s)$ that have observable training labels. As a simple illustrative example, we could have $\mathcal{S}_{tr} = \{1, \ldots, s_{tr}\}$ for some upper-bounding discrete time index s_{tr}. Moreover, for each $s \in \mathcal{S}_{tr}$, we could have a single row i added to the target table T^k, in which case $\psi_{tr}(s) = s$. Of course this setup is quite flexible, and easily accommodates a more complex evolution of $\mathcal{D}(s)$.

Additionally, we remark that an implicit assumption baked into (4), and the aforementioned designation of the training set, is that each $\boldsymbol{T}^k_{ij}(s)$ is independent of one another when conditioned on $f\left(\mathcal{D}(s) \setminus \boldsymbol{T}^k_{ij}(s); \theta\right)$. This assumption is foundational to empirical risk minimization (ERM), upon which much of the learning theory associated with fixed-sized training sets is based [92].[2] Even so, it need *not* be the case that individual rows of $\boldsymbol{T}^k(s)$ are unconditionally independent of one another.

Inference. Given some θ^* obtained by minimizing (4), at test time we are presented with new instances $\{(s,i) : i \in \psi_{te}(s), s \in \mathcal{S}_{te}\}$, from which we can compute $\hat{p}\left(\boldsymbol{T}^k_{ij}(s) \mid f\left[\mathcal{D}(s) \setminus \boldsymbol{T}^k_{ij}(s); \theta^*\right]\right)$, ideally approximating the true target distribution. We note that a transductive reduction of the above procedure naturally emerges when s is fixed across both training and testing. More generally though, as s increments forward in time, $\mathcal{D}(s)$ may undergo significant changes, such as new rows appended to $\boldsymbol{T}^k(s)$, new labels/values added to the target column $\boldsymbol{T}^k_{:j}(s)$, as well as arbitrary changes to other tables $\boldsymbol{T}^{k'}(s)$ with $k' \neq k$.

2.4 Overview of Modeling Components

While in principle the function $f(\,\cdot\,;\theta)$ from (4) can be arbitrary, practical realizations are challenging, especially relative to single-table prediction models as will be spelled out in Section 2.5. But at least for conceptual digestion, and to cover the assortment of solutions currently available in the literature, we introduce the composite functional form

$$f(\,\cdot\,;\theta) \;\coloneqq\; f_{pred} \circ f_{\mathcal{G}} \circ f_{RDB}(\,\cdot\,;\theta). \quad (5)$$

In this expression, along with the companion illustration from Figure 2, the function f_{RDB} first converts an arbitrary collection of raw tables, with missing or incomplete relational information/schema, into an RDB with all PK-FK pairs established (i.e., a known schema); details are deferred to Section 3, where the motivation for an intermediate RDB representation/abstraction will also be discussed. Next $f_{\mathcal{G}}$ converts an arbitrary RDB into a heterogeneous graph (or hypergraph) as will be detailed in Section 4. And finally, the prediction head f_{pred}, as will be fleshed out in Section 5, leverages various graph ML architectures to produce model outputs. Note that beyond powerful GNN designs, many classical forms of data augmentation or feature engineering can be recast from a graph ML perspective, even if not originally derived as such.

2.5 Challenges beyond Single-Table Modeling

Additional label leakage pathways. Label leakage occurs when features are used during model training (perhaps inadvertently)

[2] There are alternatives to ERM for making predictions specifically on RDBs, e.g., based on first-order logic [16, 39, 107]; however, for scalable, data-driven ML or deep learning solutions, ERM is a well-placed assumption.

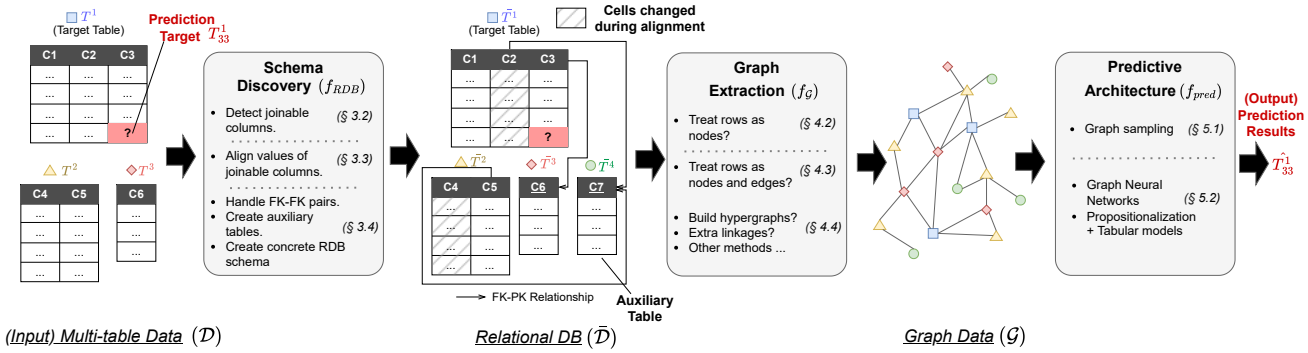

Figure 2: Overview of the generic multi-table predictive pipeline covered within this survey.

that are unlikely to be present at inference time [55]. A persistent problem even with single-table data [54], the result can be a substantial degradation in performance relative to what might otherwise be expected. But the risk of label leakage, and the difficulty of mitigation, is compounded when we move to multi-table relational data. In this setting, there now exists the possibility of ambiguous columns in auxiliary tables containing implicit label information (feature leakage) or temporal quantities measured after the target label (temporal leakage). Similarly, representative splits between training and testing may be more difficult to achieve when all data are entangled within the same evolving relational structure (group leakage). Hence care must be taken when forming features and data splits, especially w.r.t. temporal aspects [31].

Variable input size and structure. Unlike typical tabular modeling tasks involving just a single table with fixed-length features, the multi-table setting invariably requires processing sets of input features of varying size and relational context. Hence any model must either include processing steps to flatten information into a single table [13, 116], or else possess the ability to accept flexible input sets. It is with respect to the latter that graph ML techniques particularly excel, e.g., GNNs are naturally equipped to handle such inputs [100], label leakage issues notwithstanding.

More complex 3-way learning space. Of course any invocation of graph ML requires the actual existence of a graph, and it is here that another critical distinction emerges. In typical single-table regimes, a tabular model must only (either explicitly or implicitly) learn whether any given column contains useful predictive information or not, i.e., a column may or may not be included in determining final model outputs, leading to a binary selection for each column over the table. However, for any constituent table within $\mathcal{D}(s)$, the complexity expands dramatically, in part because for each column there is now a *three-way* space of possibilities: A column can either be treated as (i) a standard informative feature (as in the single table case), (ii) a garbage feature, or (iii) a relational feature dictating inter- or intra-table links (unique to multiple tables).

3 SCHEMA DISCOVERY: CONVERTING RAW MULTI-TABLE DATA TO RDBS

As mentioned in Section 2.5, one of the key challenges beyond single-table modeling is the proper handling of potential relational information contained within multiple tables. In this regard, RDBs represent a useful intermediate abstraction between raw tables on the one hand, and graphs with explicitly defined nodes and edges that reflect entity relationships on the other. Moreover, upon surveying existing work applying graph ML to tabular data, we regularly find components or model inputs tantamount to an RDB [3, 47, 107, 110]. Conceptually, the RDB creation step can be viewed as isolating all columns in the original raw tables that designate cross table and/or cross row relationships (the tertiary potential role of any column that can significantly complicate multi-table predictive tasks). Hence after the RDB is created, we only need consider which database columns contain useful predictive information or not (a binary decision, analogous to single-table tabular learning as discussed in Section 2.5).

To proceed along these lines, we first detail the core ingredients of an RDB in Section 3.1. We then describe join discovery (JD) and alignment processes in Sections 3.2 and 3.3 that facilitate RDB creation. Piecing these ingredients together to form the mapping function f_{RDB}, which transforms $\mathcal{D}(s)$ to an RDB with known schema, is then presented in Section 3.4. An illustration of this process over simple toy data is shown in Figure 3.

3.1 RDB Preliminaries

What establishes a *relational* database, as opposed to merely a collection of tables $\mathcal{D}(s)$, is that certain table columns are designated as either *primary keys* (PKs) or *foreign keys* (FKs) [36]. A column $\boldsymbol{T}^k_{:j}$ serves as a PK when each element is a unique index referencing a row of $\boldsymbol{T}^k$, such as a user ID for example. In contrast, $\boldsymbol{T}^k_{:j}$ is defined as a FK column if each $\boldsymbol{T}^k_{ij}$ corresponds with a unique PK value referencing a row in *another* table $\boldsymbol{T}^{k'}$ (generally $k' \neq k$, although this need not strictly be the case), with the only restriction being that all such indices within a given FK column must point to rows within the same table. In this way, the domain of any FK column is given by the corresponding PK column it references. Please see Figure 2 (middle) for a simple RDB example with three PK-FK pairs connecting various columns.

To notationally accommodate the above considerations, we define an RDB as some $\mathcal{D}(s)$ coupled with a set of M PK-FK pairs denoted as

$$\text{PK-FK}^m := \left\{k^m_{PK}, j^m_{PK}, k^m_{FK}, j^m_{FK}\right\}, \ \forall m = 1, \ldots, M, \tag{6}$$

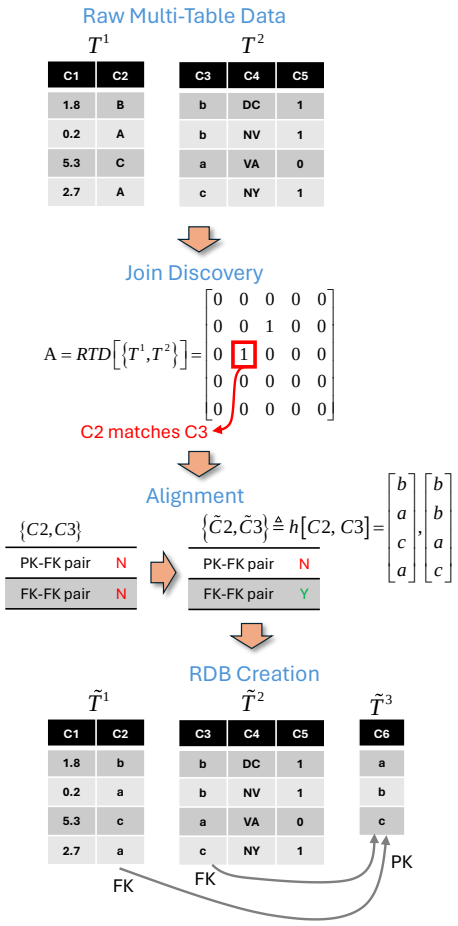

Figure 3: Process for converting raw multi-table data into an RDB with known schema. See Sections 3.2-3.4 for details.

where k^m_{PK} and j^m_{PK} denote the table and column indeces of the PK within the m-th PK-FK pair, while k^m_{FK} and j^m_{FK} are defined analogously for the corresponding FK. We also note that while each PK-FK^m is unique, either the PK or FK component in isolation need not be; this is simply because any specific PK or FK can be included within multiple different PK-FK pairs. For example, the column labeled "C7" in Figure 2 (middle) serves as a PK connected to two separate FKs.

3.2 Join Discovery

Join discovery (JD) as used herein refers to the task of finding joinable relationship between columns spanning one or more tables [20, 62], with numerous practical use cases such as feature augmentation or data cleaning. In the present context however, JD will be more narrowly leveraged to produce candidate PK-FK pairs, or as we will later see, FK-FK pairs that can be subsequently converted to two separate PK-FK pairs through the judicious introduction of additional auxiliary tables.

Mathematically, we instantiate JD as an operator designed to create what can be viewed as a form of cross-table, symmetric adjacency matrix[3] given by

$$\boldsymbol{A}(s) := JD[\mathcal{D}(s)] \in \{0,1\}^{n\times n}, \text{ with } n := \sum_k n^k(s). \tag{7}$$

By design, each upper- (or equivalently lower-) triangular element of $\boldsymbol{A}(s)$ corresponds with a unique pairing of columns drawn from $\mathcal{D}(s)$, with a value of one if they are joinable and zero otherwise (note that we also allow both columns to be drawn from the same table; more on this later). There exist a wide variety of tools for computing $\boldsymbol{A}(s)$ by exploiting a process known as fuzzy table joining [12, 69, 94]. In this regard, SOTA systems are generally based on different forms or combinations of similarity measures [19, 62, 113] applied to table columns, and increasingly, the use of language models [21, 27]. Alternatively, a skilled data scientist can at times manually construct $\boldsymbol{A}(s)$, or related graph-based quantities derived thereof, using intuition and domain knowledge [22].

3.3 Alignment

While a useful starting point, the adjacency matrix $\boldsymbol{A}(s)$ alone is not sufficient for constructing an RDB, largely because additional steps are needed to formally extract PK-FK pairs, or address what amounts to an alignment problem [46, 69, 89, 118]. Let $\rho := \{k', j', k'', j''\}$ denote a column pair, specified by column j' in table k' and column j'' in table k'', that have been joined per $\boldsymbol{A}(s)$, i.e., the associated adjacency entry is equal to one. To qualify as a PK-FK pair, it must be the case that either the values in $T^{k'}_{:j'}$ are unique and a superset of the values in $T^{k'}_{:j'}$, in which case $\{k', j'\}$ indexes the PK and $\{k'', j''\}$ the corresponding FK. Or else vice versa with the PK-FK roles reversed.[4] However, neither situation need necessarily be the case, and for any such candidate ρ there exists five relevant possibilities to consider as follows:

(a) ρ satisfies the conditions for an aligned PK-FK pair, and we treat this join as such moving forward.
(b) Neither $T^{k'}_{:j'}$ nor $T^{k''}_{:j''}$ contain all unique values; however, their respective column value domains overlap. In this case they can be viewed as forming an FK-FK pair; see Section 3.4 for details regarding how FK-FK pairs can be molded within an RDB.
(c) Let $\left\{\widetilde{T}^{k'}_{:j'}, \widetilde{T}^{k''}_{:j''}\right\} := h\left(T^{k'}_{:j'}, T^{k''}_{:j''}\right)$ denote revised columns produced by some mapping h. Furthermore, assume that either the values in $\widetilde{T}^{k'}_{:j'}$ are unique and a superset of the values in $\widetilde{T}^{k''}_{:j''}$, or vice versa. Then we update the respective tables and treat these columns as a PK-FK pair.
(d) Neither $\widetilde{T}^{k'}_{:j'}$ nor $\widetilde{T}^{k''}_{:j''}$ contain all unique values; however, their respective column value domains overlap. Then we update the respective tables and treat these columns as a FK-FK pair.
(e) If none of the above four criteria are satisfied, ρ can be optionally viewed as a spurious join and excluded from further consideration. (This could occur, for example, if the JD operator is overly aggressive in suggesting candidate joins.)

[3] Although technically this adjacency matrix induces a graph, in the present context this can be viewed as a schema-level graph, not an instance-level graph as we will later derive for making predictions. More on this in Sections 4 and 5.
[4] Note that if the values in both columns are unique and perfectly overlapping, either could be treated as a PK and it is reasonable then to just directly merge the tables.

Given the possibilities outlined above, the primary unresolved initiative is to select an appropriate alignment operator h. The basic idea here is to preserve as much as possible of the original column information, while filtering out noise or spurious differences that might otherwise disguise the existence of a PK-FK pair (or to a lesser extent, an FK-FK pair) [46, 69, 89, 118]. As a trivial example, a column of city names in one table might include the entry "New York," while in another table the abbreviation "NY" is adopted instead; a successful alignment operator would convert these both to an equivalent/shared representation. (Analogously, in Figure 3 column "C2" is shifted to lower-case to match "C3.") Once alignment is completed, we have a set of explicit PK-FK pairs, and potentially also a set of FK-FK pairs. Given these components, we discuss the final steps in forming a fully specified RDB next.

3.4 Final RDB Schema Formation

The PK-FK pairs extracted in the previous section can directly contribute to the final RDB formation. However, there remain other sources of PK-FK pairs.

Handling FK-FK pairs. Each FK-FK pair can be converted to new PK-FK pairs as follows. First, we create a new auxiliary table $T^a(s) \in \mathbb{R}^{n^a(s)\times 1}$, where $n^a(s)$ denotes the number of unique values across both columns of the FK-FK pair. Then $T^a(s)$ can be populated with these unique values, assuming the role of a new auxiliary PK, and two new PK-FK pairs are subsequently induced. See Figure 3 for a representative example of this process.

Single FK to a PK-FK pair. There is also a second approach for generating additional PK-FK pairs that has previously been adopted even in single-table prediction tasks as a means of generating cross row dependencies [44, 98]. The basic idea is to treat one or more table columns with categorical features as an isolated FK, and then create a new auxiliary PK table $T^a(s)$, with elements restricted to the unique values of the FK.

Resulting RDB Description. The final RDB with estimated schema that emerges from this process can be denoted as

$$\bar{\mathcal{D}}(s) := f_{RDB}[\mathcal{D}(s)] = \left[\left\{ \bar{T}^k(s) \right\}_{k=1}^{\bar{K}}, \left\{ \text{PK-FK}^m \right\}_{m=1}^{M} \right], \quad (8)$$

where the set of $\bar{K}$ tables includes the original K plus any additional auxiliary tables as described above. For reference, $K = 3$ and $\bar{K} = 4$ in the example from Figure 2, while $K = 2$ and $\bar{K} = 3$ in the example from Figure 3. Moreover, we use $\bar{T}^k(s)$ to reference each constituent table, given that even an original $T^k(s)$ could have been modified via the h alignment operator described in Section 3.3. And finally, we are assuming that all three candidate sources for finding PK-FK matches (namely direct, FK-FK conversion, or single FK conversion) combine to produce M pairs.

We emphasize that, although in this section we have detailed a pipeline that can in principle automate the construction of RDBs from tables, one or more of the JD, alignment, and/or final PK-FK creation steps are often handled manually in practice by experienced data scientists. Even so, given the versatility of existing tools, and the promise of further enhancements through the use of (large) language models for schema discovery and related [21, 27, 29], we expect that increasing automation is inevitable.

4 EXTRACTING GRAPHS FROM RDBS

Given an RDB $\bar{\mathcal{D}}(s)$ as described in the previous section, we now turn to the process of mapping to a heterogeneous graph, or equivalently, instantiating the $f_{\mathcal{G}}$ operator from (5).

4.1 Graph Preliminaries

A *heterogeneous graph* $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ [88] is defined by sets of node types V and edge types E such that $\mathcal{V} = \bigcup_{v\in V} \mathcal{V}^v$ and $\mathcal{E} = \bigcup_{e\in E} \mathcal{E}^e$, where $\mathcal{V}^v$ references a set of $|\mathcal{V}^v|$ nodes of type v, while $\mathcal{E}^e$ indicates a set of $|\mathcal{E}^e|$ edges of type e. Both nodes and edges can have associated features, denoted x_i^v and z_j^e for node i of type v and edge j of type e respectively. Additionally, if we allow for typed edges linking together arbitrary numbers of nodes possibly greater than two, which defines a so-called hyperedge, then $\mathcal{G}$ generalizes to a heterogeneous *hypergraph* [6, 87], a perspective that will provide useful context below. And finally, for a dynamic graph $\mathcal{G}(s)$, all of the above entities can be generalized to depend on a state variable s as before. We next consider multiple practical approaches for converting an RDB into a heterogeneous graph (or possibly hypergraph), noting that there is not as of yet any consensus on which approach is most effective on downstream predictive tasks [95].

4.2 Converting Rows to Nodes

Perhaps the most natural and intuitive way to convert an RDB $\bar{\mathcal{D}}$ to a heterogeneous graph $\mathcal{G}$ is to simply treat each row as a node, each table as a node type, and each FK-PK pair as a directed edge. Additionally, non-FK/PK column values are converted to node features assigned to the respective rows. Per this construction, row $\boldsymbol{T}^k_{i:}$ defines a node of type $v = k$. Similarly, if $\boldsymbol{T}^k_{:j}$ represents an FK column of table k that references $\boldsymbol{T}^{k'}_{:j'}$, the PK column j' of table k', then there exists an edge of type $e = kk'$ between the corresponding nodes whenever $\boldsymbol{T}^k_{ij} \rightarrow \boldsymbol{T}^{k'}_{i'j'}$ for row indices i and i'. We will refer to this graph composition as *Row2Node* for convenience. As a relatively straightforward procedure for extracting graphs from RDBs, Row2Node was proposed in [16] with ongoing application by others [31, 110, 115]. Figure 4 (top-right) visualizes the graph constructed by Row2Node from an RDB involving four tables, namely, "View," "Purchase," "Product," and "User."

4.3 Selectively Converting some Rows to Edges

As an alternative to Row2Node, we may relax the restriction that every row must be exclusively converted to a node. Instead, for tables with two FK columns, i.e., $\boldsymbol{T}^k_{:j}$ and $\boldsymbol{T}^k_{:j'}$ are both FKs with $j \neq j'$, each row can be converted to an edge connecting the corresponding rows being indexed by the FK pair. More concretely, each such $\{\boldsymbol{T}^k_{ij}, \boldsymbol{T}^k_{ij'}\}$ pair defines an edge of type $e = k'k''$ between rows of tables $\boldsymbol{T}^{k'}$ and $\boldsymbol{T}^{k''}$ as pointed to by $\boldsymbol{T}^k_{:j}$ and $\boldsymbol{T}^k_{:j'}$ respectively. The remaining columns of $\boldsymbol{T}^k$ are designated as edge features. Overall, the intuition here is simply that tables with multiple FKs can be treated as though they were natively a tabular representation of edges. This treatment has been implicitly assumed in forming past graph-based predictors [4, 112], and also previously compared head-to-head versus Row2Node [95].

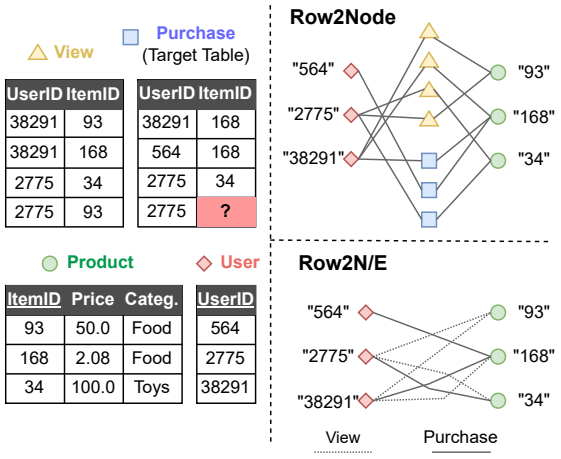

Figure 4: Illustration of Row2Node and Row2N/E graph extraction. PK columns in the tables are underlined, while the corresponding FK columns have matching header text. For Row2N/E, the dotted edges result from the "View" table, while the solid edges are produced by the "Purchase" table.

Additionally, to ensure edges exclusively connect to nodes instead of other edges as required in forming a canonical graph, we only convert rows as described above to edges if table $\boldsymbol{T}^k$ has no PK column. (If it were to have both two FK columns *and* a PK column, then a referencing FK in yet another table could lead to edge-edge connections which are disallowed by convention.) We may also expand this procedure to generate arbitrary hyperedges [6] by analogously handling tables with three or more FK columns. Overall, we refer to this conversion procedure as *Row2N/E* (short for Row-to-Node-or-Edge), as each row is now selectively treated as either a node or edge/hyperedge depending on the presence of multiple FKs. And if no table within $\tilde{\mathcal{D}}$ has multiple FKs (along with no PK column), then Row2Node and Row2N/E are equivalent.

In Figure 4, both the "View" and "Purchase" tables have two FKs ("UserID" and "ItemID" columns), and so Row2N/E becomes distinct from Row2Node. Notably, the Row2N/E graph is much simpler, with fewer hops needed to connect relevant entities.

4.4 Additional Possibilities

Within the context of GNNs applied to single tables, it has been shown that useful hypergraph edges can be created using the retrieval of similar rows to a target instance [23], and this procedure can be extended to multiple tables. Analogously, cross-row connections modulated by self-attention as proposed in [60, 85] are tantamount to introducing denser graph structure. There also exists models designed to exploit broader hypergraph structure within the context of RDBs [3, 47, 107].

5 TRAINABLE GRAPH-BASED ARCHITECTURES

The last step in operationalizing the composite pipeline from (5) involves the trainable, graph-based prediction module for computing $\hat{T}^k_{ij} := f_{pred}\left(\mathcal{G}(s) \setminus T^k_{ij}\right)$. Here the exclusion operator '\' now simply removes the node feature attributes associated with $\boldsymbol{T}^k_{ij}(s)$ from $\mathcal{G}(s)$, and $\mathcal{G}(s) = f_{\mathcal{G}} \circ f_{RGB}[\mathcal{D}(s)]$ as described in the previous two sections. Typical candidates for f_{pred} further decompose into two components: (i) a sampling operator Φ designed to exclude information in $\mathcal{G}(s)$ that is likely irrelevant to predicting a target T^k_{ij}, and (ii) a trainable, parametric function g that accepts as input a subgraph. We then arrive at

$$f_{pred}\left(\mathcal{G}(s) \setminus T^k_{ij}\right) \;=\; g\left(\Phi\left[\mathcal{G}(s) \setminus T^k_{ij}\right]; \theta\right). \tag{9}$$

We now discuss common selections for these two components in the following subsections.

5.1 Graph-based Sampling Operators

There exist a wide variety of scalable graph sampling methods for implementing Φ [9, 14, 45, 104, 108, 119]; see [72] for a representative survey. Indeed this is one of the motivations for mapping raw tables to graphs in the first place. Still, most all available graph samplers require that we specify the effective receptive field, meaning the number of hops (or in our case tables) away from the target associated with $\boldsymbol{T}^k_{ij}(s)$ from which information is collected. One reasonable choice is to match the receptive field to the RDB schema width, i.e., the maximal number of PK-FK hops needed to reach any other table from the target table. Whatever the choice though, the output of Φ will be a subgraph of $\mathcal{G}(s)$ containing the target node corresponding to row $\boldsymbol{T}^k_{i:}(s)$. Moreover, due diligence is warranted to avoid the label leakage issues mentioned in Section 2.5, and to introduce any needed temporal constraints accordingly.

Overall, provided we allow for diversity of graph extraction and sampling, then many classical multi-table data augmentation methods can be recast in this way. For example, joining a target table k with all features from tables that can be reached by a single FK-PK join can be achieved using single-hop neighbor sampling applied to $\mathcal{G}(s)$. And for one-to-many joins (i.e., many FKs pointing to a single PK in the target table) it is a common feature engineering practice to just randomly choose a single element [13]; likewise, neighbor sampling over graphs can be optionally set to achieve the equivalent [45].

5.2 Trainable Predictive Architectures

Graph Neural Networks. Message-passing GNN architectures are based on a layer-wise stack of trainable node embeddings. Specifically, for a heterogeneous graph $\mathcal{G}$ these embeddings can be computed as

$$\boldsymbol{h}^v_{i,\ell} = \pi\left(\left\{\left\{\left(\boldsymbol{h}^{v'}_{i',\ell-1}, vv'\right) : i' \in \mathcal{N}^{vv'}_i\right\} : v' \in \mathcal{N}^v\right\}, \boldsymbol{h}^v_{i,\ell-1}; \theta\right), \tag{10}$$

where $\boldsymbol{h}^v_{i,\ell}$ denotes the embedding of node i of type v at GNN layer ℓ. In this expression, $\mathcal{N}^v$ indicates the set of node types that neighbor nodes of type v, and $\mathcal{N}^{vv'}_i$ is the set of nodes of type v' that neighbor node i of type v. Moreover, we assume that there is a unique edge type $e \equiv vv'$ associated with each pair of node types (v, v'), as is typical for the case of graphs extracted from RDBs (note also that the edge type vv' is included within the inner-most set definition to differentiate each element within the outer-most set construction). Meanwhile, π is a permutation-invariant function [106] over sets (with parameters θ), acting to aggregate or fuse information from

all neighbors of connected node types at each layer. At the output layer, the embeddings produced via (10) can be applied to making node-wise predictions, which translates into predictions of target values in column $\boldsymbol{T}^k_{:j}$.

For implementing π in practice, one of the most popular special cases is the relational graph convolutional network (RGCN) architecture [83], in which case (10) simplifies to

$$\boldsymbol{h}^v_{i,\ell} \;=\; \sigma\left(\sum_{v'\in\mathcal{N}^v}\sum_{i'\in\mathcal{N}^{vv'}_i} W^{vv'}_{\ell-1}\boldsymbol{h}^{v'}_{i',\ell-1} + W^0_{\ell-1}\boldsymbol{h}^v_{i,\ell-1}\right), \qquad (11)$$

where σ is an element-wise nonlinearity such as ReLU, and each $W^{vv'}_{\ell-1}$ and $W^0_{\ell-1}$ are trainable weight matrices. Other more sophisticated alternatives include the relational graph attention (RGAT) model [7], heterogeneous graph transformers (HGT) [50], and metapath aggregated graph neural networks (MAGNN) [33].

In all cases the resulting output layer embeddings will generally depend on which approach from Section 4 is used for graph construction; similarly for prior choices in building the initial RDB in Section 4. Other GNN-like architectures specifically designed for working with RDB-based graphs (as well as some form of implicit or explicit sampling operator Φ) can be found in [3, 47, 107, 110]. There also exist GNN models integrated with gradient boosted decision trees specifically targeting tabular node features [10, 52], as are likely to appear within graphs extracted from tables.

Graph-centric Propositionalization. As a principled alternative to using GNNs for f_{pred}, *propositionalization* [63, 64, 107] represents a broad family of approaches originally designed for generating fixed-length, potentially high-dimensional feature vectors by recursively extracting and combining information spread across a database. In the present context, for each target T^k_{ij}, this process can be viewed through a graph-centric lens as the application of a sampling operator Φ (as with GNNs) followed by an aggregation step as

$$\boldsymbol{u}^k_{ij}(s) \;:=\; \text{Agg}\left(\Phi\left[\mathcal{G}(s)\setminus \boldsymbol{T}^k_{ij}(s)\right]\right) \;\in\; \mathbb{R}^d, \qquad (12)$$

where the dimension of the augmented feature vector $\boldsymbol{u}^k_{ij}$ is independent of the tuple $\{s,k,i,j\}$. There are many choices for the effective aggregation operator such as deep feature synthesis (DFS) [53] as implemented in the publicly-available Featuretools open-source framework [30], or FastProp as part of the getML package [38]. We remark that simple 1-hop table joins are a special case of (12) when Φ is limited to 1-hop neighbor sampling.

With augmented features so obtained by propositionalization, we can subsequently invoke any parameterized tabular base model to finalize the implementation of f_{pred}. Popular selections include tree-based models such as XGBoost [11], CatBoost [81], and LightGBM, as well as deep learning alternatives such as MLPs, DeepFM [43], SAINT [85], FT-Transformers [41], and open-source toolkits capable of ensembling rich combinations thereof [1, 25, 67]. Other related frameworks based on propositionalization and strong tabular base models can be found in [13, 34, 65, 71]. Overall, while propositionalization mostly dominates ML solutions on RDBs and multi-table data thus far, more recent GNN-based alternatives are making significant headway [47, 107], especially on multi-table data with more complex intrinsic schema [95, 115].

Bridging GNNs and Propositionalization. Although their origin stories and formulations often differ, strong dimensions of overlap remain between GNN models and propositionalization-based approaches to tabular data. We highlight a few such connections as follows:

- There exists a sub-class of GNN models based on a propagate-and-predict architecture. The basic idea here is to perform parameter-free message passing and aggregation as a one-time pre-processing step (the so-called propagation), followed by prediction using any traditional deep neural network module. Representative examples include SIGN [32], SGC [97], GAMLP [114], NARS [105], and some variants of TWIRLS [102]. In all cases, these architectures can be directly viewed as instantiations of propositionalization that have been reframed within the narrative of GNNs.
- Propositionalization via DFS relies on the concatenation of various simple aggregators such as *mean*, *min*, and *max* across tables joined by PK-FK pairs. Analogously, GNN architectures based on the principal neighborhood aggregation (PNA) framework [15] exploit the exact same notion of concatenated aggregation.
- A common heuristic in dealing with high cardinality categorical columns within tabular data is to replace the original values with so-called *target encodings* [79], such as the mean or mode of the target label over every row/instance with the same value. An analogous strategy can be implicitly incorporated within various GNN architectures [84, 96] that include label propagation variants as part of the forward pass computation (10).

6 DISCUSSION AND FUTURE TRENDS

Throughout this survey, we have presented a pathway from collections of tables in an unstructured data repository, through the intermediate abstraction of an RDB, and on to powerful graph-centric models reflecting estimated relationships between tables to predict unknown quantities of interest. Along the way we have detailed available tools and techniques that can in principle be combined to reduce the necessity for manual intervention. In this regard, we expect further trending towards automation, with the increasing possibility of end-to-end systems analogous to what currently exists for single-table data (e.g., open-source tools such as [1, 67]). And as alluded to in Section 3.4, the emergence of large language models is likely to expedite this transition by tackling schema discovery and affiliated tabular challenges [21, 27, 29].

Even so, in some respects advances in tabular and relational modeling still lag what can be observed in computer vision and natural language processing fields, where pre-trained foundation models spanning wide-ranging application scenarios dominate the landscape. That being said, there do exist recent preliminary attempts to apply similar principles to graph [70, 80] and tabular [101] prediction problems. However, one notable impediment to further progress with relational data in particular is the lack of sufficiently diverse, large-scale datasets as are generally required for training. Fortunately though, new relational benchmarks are beginning to appear [31, 95, 115], and the promise of pre-training with purely synthetic data is also gaining orthogonal traction [48].

REFERENCES

[1] AutoGluon 2023. https://auto.gluon.ai/scoredebugweight/tutorials/tabular_prediction/index.html

[2] Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Transformers for tabular data representation: A survey of models and applications. *Transactions of the Association for Computational Linguistics* 11 (2023), 227–249.

[3] Jinze Bai, Jialin Wang, Zhao Li, Donghui Ding, Ji Zhang, and Jun Gao. 2021. ATJ-Net: Auto-Table-Join Network for Automatic Learning on Relational Databases. In *Proceedings of the Web Conference 2021*. 1540–1551.

[4] Rianne van den Berg, Thomas N Kipf, and Max Welling. 2017. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263* (2017).

[5] Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. 2022. Deep neural networks and tabular data: A survey. *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[6] Alain Bretto. 2013. Hypergraph theory. *An introduction. Mathematical Engineering. Cham: Springer* 1 (2013).

[7] Dan Busbridge, Dane Sherburn, Pietro Cavallo, and Nils Y Hammerla. 2019. Relational graph attention networks. *arXiv preprint arXiv:1904.05811* (2019).

[8] Qi Cao, Huawei Shen, Jinhua Gao, Bingzheng Wei, and Xueqi Cheng. 2020. Popularity prediction on social platforms with coupled graph neural networks. In *Proceedings of the 13th international conference on web search and data mining*. 70–78.

[9] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247* (2018).

[10] Jiuhai Chen, Jonas Mueller, Vassilis N. Ioannidis, Soji Adeshina, Yangkun Wang, Tom Goldstein, and David Wipf. 2022. Does your graph need a confidence boost? Convergent boosted smoothing on graphs with tabular node features. In *International Conference on Learning Representations*.

[11] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[12] Zhimin Chen, Yue Wang, Vivek Narasayya, and Surajit Chaudhuri. 2019. Customizable and scalable fuzzy join for big data. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2106–2117.

[13] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *Proc. VLDB Endow.* 13, 9 (may 2020), 1373–1387. https://doi.org/10.14778/3397230.3397235

[14] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*. 257–266.

[15] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. 2020. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems* 33 (2020), 13260–13271.

[16] Milan Cvitkovic. 2020. Supervised learning on relational databases with graph neural networks. *arXiv preprint arXiv:2002.02046* (2020).

[17] Diginetica 2016. https://competitions.codalab.org/competitions/11161

[18] DMDave, Todd B, and Will Cukierski. 2014. Acquire Valued Shoppers Challenge. https://kaggle.com/competitions/acquire-valued-shoppers-challenge

[19] Hong-Hai Do and Erhard Rahm. 2002. COMA—a system for flexible combination of schema matching approaches. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*. Elsevier, 610–621.

[20] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient joinable table discovery in data lakes: A high-dimensional similarity-based approach. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 456–467.

[21] Yuyang Dong, Chuan Xiao, Takuma Nozawa, Masafumi Enomoto, and Masafumi Oyamada. 2022. DeepJoin: Joinable Table Discovery with Pre-trained Language Models. *arXiv preprint arXiv:2212.07588* (2022).

[22] Yingtong Dou, Zhiwei Liu, Li Sun, Yutong Deng, Hao Peng, and Philip S Yu. 2020. Enhancing graph neural network-based fraud detectors against camouflaged fraudsters. In *Proceedings of the 29th ACM international conference on information & knowledge management*. 315–324.

[23] Kounianhua Du, Weinan Zhang, Ruiwen Zhou, Yangkun Wang, Xilong Zhao, Jiarui Jin, Quan Gan, Zheng Zhang, and David P Wipf. 2022. Learning Enhanced Representation for Tabular Data via Neighborhood Propagation. *Advances in Neural Information Processing Systems* 35 (2022), 16373–16384.

[24] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. 2023. Benchmarking graph neural networks. *Journal of Machine Learning Research* 24, 43 (2023), 1–48.

[25] Nick Erickson, Jonas Mueller, Alexander Shirkov, Hang Zhang, Pedro Larroy, Mu Li, and Alexander Smola. 2020. Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505* (2020).

[26] Grace Fan, Jin Wang, Yuliang Li, and Renée J Miller. 2023. Table discovery in data lakes: State-of-the-art and future directions. In *Companion of the 2023 International Conference on Management of Data*. 69–75.

[27] Grace Fan, Jin Wang, Yuliang Li, Dan Zhang, and Renée J. Miller. 2023. Semantics-Aware Dataset Discovery from Data Lakes with Contextualized Column-Based Representation Learning. *Proceedings of the VLDB Endowment* 16, 7 (2023), 1726–1739.

[28] Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. 2019. Graph neural networks for social recommendation. In *The world wide web conference*. 417–426.

[29] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. 2024. Large Language Models on Tabular Data–A Survey. *arXiv preprint arXiv:2402.17944* (2024).

[30] Featuretools 2023. https://www.featuretools.com/

[31] Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. 2023. Relational Deep Learning: Graph Representation Learning on Relational Databases. *arXiv preprint arXiv:2312.04615* (2023).

[32] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. 2020. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198* (2020).

[33] Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. 2020. Magnn: Metapath aggregated graph neural network for heterogeneous graph embedding. In *Proceedings of the web conference 2020*. 2331–2341.

[34] Sainyam Galhotra, Yue Gong, and Raul Castro Fernandez. 2023. METAM: Goal-Oriented Data Discovery. *arXiv preprint arXiv:2304.09068* (2023).

[35] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, et al. 2023. A survey of graph neural networks for recommender systems: Challenges, methods, and directions. *ACM Transactions on Recommender Systems* 1, 1 (2023), 1–51.

[36] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. 2009. *Database Systems: The Complete Book*. Prentice Hall.

[37] Andrew Gelman, John Carlin, Hal Stern, and Donald Rubin. 1995. *Bayesian Data Analysis*. Chapman and Hall.

[38] getML 2023. https://www.getml.com/

[39] Lise Getoor and Ben Taskar. 2007. *Introduction to statistical relational learning*. MIT press.

[40] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. 2017. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*. PMLR, 1263–1272.

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

[42] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. 2022. Why do tree-based models still outperform deep learning on typical tabular data? *Advances in neural information processing systems* 35 (2022), 507–520.

[43] Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. 2017. DeepFM: a factorization-machine based neural network for CTR prediction. *arXiv preprint arXiv:1703.04247* (2017).

[44] Xiawei Guo, Yuhan Quan, Huan Zhao, Quanming Yao, Yong Li, and Weiwei Tu. 2021. Tabgnn: Multiplex graph neural network for tabular data prediction. *arXiv preprint arXiv:2108.09127* (2021).

[45] Will Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems* 30 (2017).

[46] Yeye He, Kris Ganjam, and Xu Chu. 2015. Sema-join: joining semantically-related tables using big table corpora. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1358–1369.

[47] Benjamin Hilprecht, Kristian Kersting, and Carsten Binnig. 2023. SPARE: A Single-Pass Neural Model for Relational Databases. *arXiv preprint arXiv:2310.13581* (2023).

[48] Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. 2022. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848* (2022).

[49] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. 2020. Open graph benchmark: Datasets for machine learning on graphs. *Advances in Neural Information Processing Systems* 33 (2020), 22118–22133.

[50] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of the web conference 2020*. 2704–2710.

[51] Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. 2023. Temporal graph benchmark for machine learning on temporal graphs. *arXiv preprint arXiv:2307.01026* (2023).

[52] Sergei Ivanov and Liudmila Prokhorenkova. 2021. Boost then convolve: Gradient boosting meets graph neural networks. *arXiv preprint arXiv:2101.08543* (2021).

[53] James Max Kanter and Kalyan Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE international conference on data science and advanced analytics (DSAA)*. IEEE, 1–10.

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

[55] Shachar Kaufman, Saharon Rosset, Claudia Perlich, and Ori Stitelman. 2012. Leakage in data mining: Formulation, detection, and avoidance. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6, 4 (2012), 1–21.

[56] Steven M. Kearnes, Kevin McCloskey, Marc Berndl, Vijay S. Pande, and Patrick Riley. 2016. Molecular graph convolutions: moving beyond fingerprints. *J. Comput. Aided Mol. Des.* 30, 8 (2016), 595–608.

[57] Arpandeep Khatua, Vikram Sharma Mailthody, Bhagyashree Taleka, Tengfei Ma, Xiang Song, and Wen-mei Hwu. 2023. IGB: Addressing The Gaps In Labeling, Features, Heterogeneity, and Size of Public Graph Datasets for Deep Learning Research. *arXiv preprint arXiv:2302.13522* (2023).

[58] Hwan Kim, Byung Suk Lee, Won-Yong Shin, and Sungsu Lim. 2022. Graph anomaly detection with graph neural networks: Current status and challenges. *IEEE Access* 10 (2022), 111820–111829.

[59] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations* (Palais des Congrès Neptune, Toulon, France) *(ICLR '17)*. Palais des Congrès Neptune, Toulon, France. https://openreview.net/forum?id=SJU4ayYgl

[60] Jannik Kossen, Neil Band, Clare Lyle, Aidan N Gomez, Thomas Rainforth, and Yarin Gal. 2021. Self-attention between datapoints: Going beyond individual input-output pairs in deep learning. *Advances in Neural Information Processing Systems* 34 (2021), 28742–28756.

[61] Christos Koutras, Kyriakos Psarakis, George Siachamis, Andra Ionescu, Marios Fragkoulis, Angela Bonifati, and Asterios Katsifodimos. 2021. Valentine in action: matching tabular data at scale. *Proceedings of the VLDB Endowment* 14, 12 (2021), 2871–2874.

[62] Christos Koutras, Jiani Zhang, Xiao Qin, Chuan Lei, Vasileios Ioannidis, Christos Faloutsos, George Karypis, and Asterios Katsifodimos. 2024. OmniMatch: Effective Self-Supervised Any-Join Discovery in Tabular Data Repositories. *arXiv preprint arXiv:2403.07653* (2024).

[63] Stefan Kramer and C Bessiere. 2020. A brief history of learning symbolic higher-level representations from data (and a curious look forward).. In *IJCAI*. 4868–4876.

[64] Stefan Kramer, Nada Lavrač, and Peter Flach. 2001. *Propositionalization Approaches to Relational Data Mining*. Springer Berlin Heidelberg, Berlin, Heidelberg, 262–291. https://doi.org/10.1007/978-3-662-04599-2_11

[65] Arun Kumar, Jeffrey Naughton, Jignesh M Patel, and Xiaojin Zhu. 2016. To join or not to join? thinking twice about joins before feature selection. In *Proceedings of the 2016 International Conference on Management of Data*. 19–34.

[66] Sanjay Kumar, Abhishek Mallik, Anavi Khetarpal, and Bhawani Sankar Panda. 2022. Influence maximization in social networks using graph embedding and graph neural network. *Information Sciences* 607 (2022), 1617–1636.

[67] Erin LeDell and Sebastien Poirier. 2020. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, Vol. 2020. ICML.

[68] Cheng-Te Li, Yu-Che Tsai, Chih-Yao Chen, and Jay Chiehen Liao. 2024. Graph Neural Networks for Tabular Data Learning: A Survey with Taxonomy and Directions. *arXiv preprint arXiv:2401.02143* (2024).

[69] Peng Li, Xiang Cheng, Xu Chu, Yeye He, and Surajit Chaudhuri. 2021. Auto-fuzzyjoin: Auto-program fuzzy similarity joins without labeled examples. In *Proceedings of the 2021 international conference on management of data*. 1064–1076.

[70] Hao Liu, Jiarui Feng, Lecheng Kong, Ningyue Liang, Dacheng Tao, Yixin Chen, and Muhan Zhang. 2023. One for all: Towards training one graph model for all classification tasks. *arXiv preprint arXiv:2310.00149* (2023).

[71] Jiabin Liu, Chengliang Chai, Yuyu Luo, Yin Lou, Jianhua Feng, and Nan Tang. 2022. Feature augmentation with reinforcement learning. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 3360–3372.

[72] Xin Liu, Mingyu Yan, Lei Deng, Guoqi Li, Xiaochun Ye, and Dongrui Fan. 2021. Sampling methods for efficient training of graph convolutional networks: A survey. *IEEE/CAA Journal of Automatica Sinica* 9, 2 (2021), 205–234.

[73] Zhiwei Liu, Yingtong Dou, Philip S Yu, Yutong Deng, and Hao Peng. 2020. Alleviating the inconsistency problem of applying graph neural network to fraud detection. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*. 1569–1572.

[74] Sitao Luan, Chenqing Hua, Minkai Xu, Qincheng Lu, Jiaqi Zhu, Xiao-Wen Chang, Jie Fu, Jure Leskovec, and Doina Precup. 2024. When Do Graph Neural Networks Help with Node Classification? Investigating the Homophily Principle on Node Distinguishability. *Advances in Neural Information Processing Systems* 36 (2024).

[75] Qingsong Lv, Ming Ding, Qiang Liu, Yuxiang Chen, Wenzheng Feng, Siming He, Chang Zhou, Jianguo Jiang, Yuxiao Dong, and Jie Tang. 2021. Are we really making much progress? revisiting, benchmarking and refining heterogeneous graph neural networks. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*. 1150–1160.

[76] mjkistler, Ran Locar, Ronny Lempel, RoySassonOB, Rwagner, and Will Cukierski. 2016. Outbrain Click Prediction. https://kaggle.com/competitions/outbrain-click-prediction

[77] Jan Motl and Oliver Schulte. 2015. The CTU prague relational learning repository. *arXiv preprint arXiv:1511.03086* (2015).

[78] Jianmo Ni, Jiacheng Li, and Julian McAuley. 2019. Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (Eds.). Association for Computational Linguistics, Hong Kong, China, 188–197. https://doi.org/10.18653/v1/D19-1018

[79] Florian Pargent, Florian Pfisterer, Janek Thomas, and Bernd Bischl. 2022. Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features. *Computational Statistics* 37, 5 (2022), 2671–2692.

[80] Bryan Perozzi, Bahare Fatemi, Dustin Zelle, Anton Tsitsulin, Mehran Kazemi, Rami Al-Rfou, and Jonathan Halcrow. 2024. Let Your Graph Do the Talking: Encoding Structured Data for LLMs. *arXiv preprint arXiv:2402.05862* (2024).

[81] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: Unbiased boosting with categorical features. *Advances in neural information processing systems* 31 (2018).

[82] Research Industry Network 2023. https://linkedin.com/pulse/2031-relational-database-management/

[83] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.

[84] Yunsheng Shi, Zhengjie Huang, Shikun Feng, Hui Zhong, Wenjin Wang, and Yu Sun. 2020. Masked label prediction: Unified message passing model for semi-supervised classification. *arXiv preprint arXiv:2009.03509* (2020).

[85] Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C Bayan Bruss, and Tom Goldstein. 2021. Saint: Improved neural networks for tabular data via row attention and contrastive pre-training. *arXiv preprint arXiv:2106.01342* (2021).

[86] StackExchange Data Explorer [n. d.]. https://data.stackexchange.com/

[87] Xiangguo Sun, Hongzhi Yin, Bo Liu, Hongxu Chen, Jiuxin Cao, Yingxia Shao, and Nguyen Quoc Viet Hung. 2021. Heterogeneous hypergraph embedding for graph classification. In *Proceedings of the 14th ACM international conference on web search and data mining*. 725–733.

[88] Yizhou Sun and Jiawei Han. 2013. Mining heterogeneous information networks: a structural analysis approach. *ACM SIGKDD Explorations Newsletter* 14, 2 (2013), 20–28.

[89] Sahaana Suri, Ihab F Ilyas, Christopher Ré, and Theodoros Rekatsinas. 2021. Ember: No-Code Context Enrichment via similarity-based keyless joins. *arXiv preprint arXiv:2106.01501* (2021).

[90] Qiaoyu Tan, Ninghao Liu, and Xia Hu. 2019. Deep representation learning for social network analysis. *Frontiers in big Data* 2 (2019), 2.

[91] Jianheng Tang, Jiajin Li, Ziqi Gao, and Jia Li. 2022. Rethinking graph neural networks for anomaly detection. In *International Conference on Machine Learning*. PMLR, 21076–21089.

[92] Vladimir Vapnik. 1991. Principles of risk minimization for learning theory. *Advances in Neural Information Processing systems* 4 (1991).

[93] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, Vancouver, BC, Canada. https://openreview.net/forum?id=rJXMpikCZ

[94] Jiannan Wang, Guoliang Li, and Jianhua Fe. 2011. Fast-join: An efficient method for fuzzy token matching based string similarity join. In *2011 IEEE 27th International Conference on Data Engineering*. IEEE, 458–469.

[95] Minjie Wang, Quan Gan, David Wipf, Zhenkun Cai, Jiahang Li, Ning Li, Mao Zunyao, Yakun Song, Jianheng Tang, Yanbo Wang, Han Zhang, Zhang Yanlin, Zizhao Zhang, Yang Guang, Chuan Lei, Xiao Qin, Muhan Zhang, Weinan Zhang, Christos Faloutsos, and Zheng Zhang. 2024. 4DBInfer: A 4D Benchmarking Toolbox for Graph-Centric Predictive Modeling on Relational DBs. *Under Review* (2024).

[96] Yangkun Wang, Jiarui Jin, Weinan Zhang, Yong Yu, Zheng Zhang, and David Wipf. 2021. Bag of tricks for node classification with graph neural networks. *arXiv preprint arXiv:2103.13355* (2021).

[97] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. 2019. Simplifying graph convolutional networks. In *International Conference on Machine Learning*. 6861–6871.

[98] Qitian Wu, Chenxiao Yang, and Junchi Yan. 2021. Towards open-world feature extrapolation: An inductive graph learning approach. *Advances in Neural Information Processing Systems* 34 (2021), 19435–19447.

[99] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. 2022. Graph neural networks in recommender systems: a survey. *Comput. Surveys* 55, 5 (2022), 1–37.

[100] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. 2020. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems* 32, 1 (2020), 4–24.

[101] Jiahuan Yan, Bo Zheng, Hongxia Xu, Yiheng Zhu, Danny Chen, Jimeng Sun, Jian Wu, and Jintai Chen. 2024. Making Pre-trained Language Models Great on Tabular Prediction. *arXiv preprint arXiv:2403.01841* (2024).
[102] Yongyi Yang, Tang Liu, Yangkun Wang, Jinjing Zhou, Quan Gan, Zhewei Wei, Zheng Zhang, Zengfeng Huang, and David Wipf. 2021. Graph Neural Networks Inspired by Classical Iterative Algorithms. In *International Conference on Machine Learning*.
[103] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 974–983.
[104] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. 2018. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 974–983.
[105] Lingfan Yu, Jiajun Shen, Jinyang Li, and Adam Lerer. 2020. Scalable graph neural networks for heterogeneous graphs. *arXiv preprint arXiv:2011.09679* (2020).
[106] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. *Advances in neural information processing systems* 30 (2017).
[107] Lukáš Zahradník, Jan Neumann, and Gustav Šír. 2023. A deep learning blueprint for relational databases. In *NeurIPS 2023 Second Table Representation Learning Workshop*.
[108] Hanqing Zeng, Muhan Zhang, Yinglong Xia, Ajitesh Srivastava, Andrey Malevich, Rajgopal Kannan, Viktor Prasanna, Long Jin, and Ren Chen. 2021. Decoupling the Depth and Scope of Graph Neural Networks. *Advances in Neural Information Processing Systems* 34 (2021).
[109] Ge Zhang, Zhao Li, Jiaming Huang, Jia Wu, Chuan Zhou, Jian Yang, and Jianliang Gao. 2022. efraudcom: An e-commerce fraud detection system via competitive graph neural networks. *ACM Transactions on Information Systems (TOIS)* 40, 3 (2022), 1–29.
[110] Han Zhang, Quan Gan, David Wipf, and Weinan Zhang. 2023. Gfs: Graph-based feature synthesis for prediction over relational databases. *arXiv preprint arXiv:2312.02037* (2023).
[111] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in neural information processing systems* 31 (2018).
[112] Muhan Zhang and Yixin Chen. 2019. Inductive matrix completion based on graph neural networks. *arXiv preprint arXiv:1904.12058* (2019).
[113] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. 2011. Automatic discovery of attributes in relational databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 109–120.
[114] Wentao Zhang, Ziqi Yin, Zeang Sheng, Yang Li, Wen Ouyang, Xiaosen Li, Yangyu Tao, Zhi Yang, and Bin Cui. 2022. Graph attention multi-layer perceptron. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4560–4570.
[115] Zizhao Zhang, Yi Yang, Lutong Zou, He Wen, Tao Feng, and Jiaxuan You. 2023. RDBench: ML Benchmark for Relational Databases. *arXiv preprint arXiv:2310.16837* (2023).
[116] Zixuan Zhao and Raul Castro Fernandez. 2022. Leva: Boosting machine learning performance with relational embedding data augmentation. In *Proceedings of the 2022 International Conference on Management of Data*. 1504–1517.
[117] Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. 2020. Graph neural networks: A review of methods and applications. *AI open* 1 (2020), 57–81.
[118] Erkang Zhu, Yeye He, and Surajit Chaudhuri. 2017. Auto-join: Joining tables by leveraging transformations. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1034–1045.
[119] Difan Zou, Ziniu Hu, Yewen Wang, Song Jiang, Yizhou Sun, and Quanquan Gu. 2019. Layer-dependent importance sampling for training deep and large graph convolutional networks. *Advances in Neural Information Processing Systems* 32 (2019).
[120] Roman Zykov, Noskov Artem, and Anokhin Alexander. 2022. Retailrocket recommender system dataset. https://doi.org/10.34740/KAGGLE/DSV/4471234

A ACKNOWLEDGEMENTS

Special thanks to Christos Koutras for helpful suggestions regarding this manuscript.