The Role of Feature Interactions in Graph-based Tabular Deep Learning

Anonymous authors
Paper under double-blind review

Abstract

Accurate predictions on tabular data rely on capturing complex, dataset-specific feature interactions. Attention-based methods and graph neural networks, referred to as graph-based tabular deep learning (GTDL), aim to improve predictions by modeling these interactions as a graph. In this work, we analyze how these methods model the feature interactions. Current GTDL approaches primarily focus on optimizing predictive accuracy, often neglecting the accurate modeling of the underlying graph structure. Using synthetic datasets with known ground-truth graph structures, we find that current GTDL methods fail to recover meaningful feature interactions, as their edge recovery is close to random. This suggests that the attention mechanism and message-passing schemes used in GTDL do not effectively capture feature interactions. Furthermore, when we impose the true interaction structure, we find that the predictive accuracy improves. This highlights the need for GTDL methods to prioritize accurate modeling of the graph structure, as it leads to better predictions.

1 Introduction

Deep learning has achieved remarkable success in domains such as natural language processing and computer vision. On tabular data, however, deep learning methods still struggle to compete against traditional, tree-based machine learning methods (Grinsztajn et al., 2022; McElfresh et al., 2024). Although recent advances in tabular deep learning occasionally surpass these baselines on select benchmarks (e.g., Gorishniy et al. (2021); Hollmann et al. (2025)), no deep learning method has yet demonstrated consistent superiority across datasets and evaluation settings (Grinsztajn et al., 2022; McElfresh et al., 2024).

Tabular data is characterized by the heterogeneous nature of its features: each feature often encodes distinct semantics, and relationships among features (or feature interactions) can be complex, indirect, and dataset-specific. By modeling the feature interactions, one incorporates the *inductive bias* (domain-specific principles embedded into the model's architecture (Goyal & Bengio, 2022; Battaglia et al., 2018; Romero Guzman, 2024)) that features interact with each other differently. With 'modeling' we mean that the network has, by design, separate parameters for each feature interaction. Using inductive biases has proven to be important for success in other fields of deep learning. For example, convolutional neural networks (CNNs) achieve sample efficiency and robustness in computer vision by encoding translational invariance (LeCun et al., 1989; Fukushima, 1980), and transformers excel in natural language processing using attention-based mechanisms to capture sequential and contextual relationships (Vaswani et al., 2017).

Modeling this inductive bias of feature interactions comes naturally in the form of a graph, where the nodes represent features and the edges their interactions. Probabilistic graphical models (PGMs) have a rich history in statistics, providing a framework to model multivariate dependencies (Lauritzen, 1996). These methods excel at robustly describing the graph structure while enabling predictions, yet they lack the ability to model complex nonlinear relationships that deep learning can provide. Graph-based tabular deep learning (GTDL) methods aim to merge the expressive power of deep learning with graph-structured feature representations. *Feature* graph neural networks (GNNs), reviewed by Li et al. (2024), are GNNs focused on tabular data,

having the features as nodes and the feature interactions as edges.[1] However, how these feature interactions are modeled within these networks has not been extensively studied or evaluated. These methods do not evaluate explicitly whether their learned feature interactions accurately correspond to meaningful relationships in the data.

Existing GTDL methods (e.g., Li et al. (2019a); Yan et al. (2023); Zheng et al. (2023); Villaizán-Vallelado et al. (2024); Ye et al. (2024); Zhou et al. (2022)) typically evaluate the learned graph structure only qualitatively, as real-world datasets rarely include ground-truth feature interaction graphs. Their training loss is tied to predictive performance, providing no incentive to ensure accuracy or meaningfulness in the learned graph structure. As a result, the adjacency matrix may reflect optimization artifacts rather than genuine feature interactions, as sketched in Figure 1. This emphasis on predictive metrics over structural fidelity constrains interpretability. Overall, there remains a lack of systematic techniques for both validating learned feature interactions and guiding learning with prior knowledge.

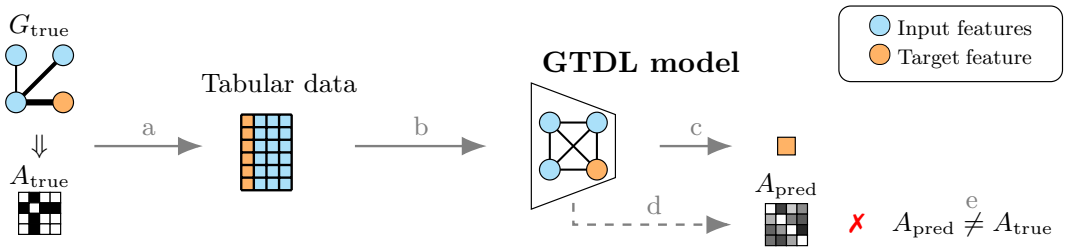

Figure 1: (a) A true underlying graph structure generates tabular data. (b, c) Only the data is used to train a GTDL model (using a fully connected graph) to predict the target feature. (d) After training, the learned graph structure is extracted from the model. (e) When this predicted graph structure is compared to the true graph structure, we find that they are not similar for existing GTDL methods.

In this work, we analyze whether existing GTDL methods learn meaningful feature interactions. Do GTDL methods learn an accurate graph structure when being trained to predict a target feature? Conversely, does an exclusive focus on predictive accuracy lead to spurious interactions rather than genuine feature dependencies, thereby undermining robustness, generalization, and explainability? When the graph structure is modeled correctly, does the predictive performance of GTDL methods improve?

To support our analysis, we take the following steps. In Section 2, we review the existing literature on GTDL methods and identify their limitations in the evaluation and validation of learned feature interactions. In Section 3, we introduce a framework to systematically evaluate how well predictive GTDL methods learn the graph structure. The two key parts of this framework are (i) synthetic datasets with known ground-truth graphs, and (ii) a metric to quantitatively assess the accuracy of learned graphs. In Section 4, we discuss the results of controlled experiments to showcase the framework. Specifically, we show that existing GTDL methods fail to recover meaningful feature interactions, and that enforcing the true interaction structure improves predictive performance. In Section 5, we give a concluding discussion and propose future research directions.

2 GTDL methods and their feature interactions

After formalizing the problem context, we organize this section around methods that model feature interactions in tabular data through feature graphs. First, review attention-based methods and GNNs for tabular data, and discuss how both are interpreted as GTDL methods. Next, we relate GTDL to PGMs, which serve as a principled baseline for learning conditional independence structure. Lastly, we investigate methods that may appear related at first glance but, upon closer examination, are not directly pertinent to this study.

[1]This categorization of feature graphs contrasts with instance graphs, where nodes represent instances (rows) and edges capture relationships between those instances.

Problem setting and notation. A full tabular dataset $D = [x \parallel y] \in \mathbb{R}^{n \times p}$ consists in a traditional supervised setting of input features $x \in \mathbb{R}^{n \times (p-1)}$ and a target feature $y \in \mathbb{R}^{n \times 1}$, with n the number of samples, p the number of features and $\parallel$ indicating concatenation. When we refer to 'features', we mean both input and target features. Features could be either numerical or categorical.

The features and their interactions can be represented as an undirected graph $G = (V, E)$, where V is the set of nodes and E the set of edges, such that the number of nodes $|V| = p$. A binary symmetric adjacency matrix A_{true} describes the true graph structure. The absence of an edge between two nodes indicates the conditional independence of these two nodes conditioned on all other nodes. (This is different from directed causal graphs, where the presence of an edge indicates a direct causal effect between two nodes.) From trained GTDL methods, we can extract a weighted adjacency matrix $A \in \mathbb{R}^{p \times p}$, where $0 \leq A_{ij} \leq 1$ indicates the strength of the interaction between features i and j, with $i, j \in \{1, \ldots, p\}$. As the interaction from feature i to j could be different from the interaction from feature j to i, the weighted adjacency matrix is not necessarily symmetric: While A_{ij} can differ from A_{ji}, they should be symmetric in support (i.e., they share the same sparsity pattern). Specifically, if $A_{\text{true},ij} = 0$, then it should be that $A_{ij}, A_{ji} = 0$. Conversely, if $A_{\text{true},ij} = 1$, then it should be that $A_{ij}, A_{ji} > 0$.

The main task is to predict the target feature y given the input features x. Correct modeling of the feature interactions could improve this prediction, including both target-input interactions (relationships between y and each x_i) and input-input interactions (relationships between pairs of input features x_i and x_j). The strength of feature interactions between two nodes should be (close to) zero when these nodes are conditionally independent.

2.1 Attention-based methods

Due to the success of the transformer architecture (Vaswani et al., 2017), most recent tabular deep learning methods are attention-based (Arik & Pfister, 2020; Huang et al., 2020; Somepalli et al., 2021; Kossen et al., 2021; Gorishniy et al., 2021; Hollmann et al., 2025), from which FT-Transformer (Gorishniy et al., 2021) has been established as a popular baseline. All of these methods are based on multi-head self-attention, with the attention module

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V = aV,$$

with Q, K and V the query, key and value matrices, respectively. We use lower case a to denote the attention map to distinguish it from the adjacency matrix A.

In most works, the attention map a is of size $p \times p$ if a target token (or CLS, 'classification' or 'output' token (Devlin et al., 2019)) is appended. If the attention map is equal to the size of the total number of features (or the number of input features), it can be used for interpretation and explaining the feature interactions. This approach is popular in natural language, with BertViz (Vig, 2019) being used to interpret how the model assigns weights to different tokens. We note that when the attention map is averaged over the samples, heads, and layers, it can be interpreted as a weighted adjacency matrix. Details how we extract the attention map as a weighted adjacency matrix are provided in Appendix A. Therefore, we refer to such attention-based methods as *implicit* GTDL methods. They do not explicitly model the feature interactions, but due to the nature of the attention map, they do model the graph structure implicitly.

This notion, that the attention map can be interpreted as the learned graph structure of tabular data, has not been thoroughly discussed in the literature. However, most methods, as listed in Table 1, use the attention map for interpretability. TabNet (Arik & Pfister, 2020) reports the attention map of synthetic datasets as visualizations and notes how irrelevant features are ignored in the attention map. SAINT (Somepalli et al., 2021), although designed for tabular data, reports the attention map of MNIST and discusses that the visualization of the attention map is similar to the ground-truth image. FT-Transformer (Gorishniy et al., 2021) interprets the attention map as feature importances, and shows for some real-world datasets that the attention map has a high rank correlation with integrated gradients (Sundararajan et al., 2017), a method to measure feature importance.

Table 1: GTDL methods evaluate the feature interaction only qualitatively, typically with a visualization of the attention map or adjacency matrix. Size denotes the length of square attention map a or adjacency matrix A (e.g., size p means $A \in \mathbb{R}^{p \times p}$).

Model	Reference	Size	Feature interaction evaluation
Attention-based			
FT-Transformer	Gorishniy et al. (2021)	p	Correlation with feature importance
TabNet	Arik & Pfister (2020)	p	Visual of synthetic dataset
SAINT	Somepalli et al. (2021)	p	Visual of MNIST
Graph neural network			
FiGNN	Li et al. (2019a)	$p-1$	Visual of real-world dataset
T2G-Former	Yan et al. (2023)	p	Visual of real-world datasets
DRSA-Net	Zheng et al. (2023)	$p-1$	Visual of real-world dataset
INCE	Villaizán-Vallelado et al. (2024)	p	Visual of real-world dataset
MPCFIN	Ye et al. (2024)	p	Visual of real-world datasets
Table2Graph	Zhou et al. (2022)	$p-1$	Visual synthetic dataset

2.2 Graph neural networks

GNNs operate directly on graph-structured data by propagating information between connected nodes (Zhou et al., 2021). Feature GNNs apply this paradigm to tabular data, modeling each feature as a node and explicitly learning feature interactions through message passing (Li et al., 2024). Because these methods use a graph structure by design, we refer to them as *explicit* GTDL methods, contrary to attention-based methods that model the graph structure implicitly. The explicit GTDL methods are initialized with a fully connected graph and learn the weighted adjacency matrix.

FiGNN (Li et al., 2019a) uses a feature graph to explicitly model the separate feature interactions. T2G-Former (Yan et al., 2023) adapts the transformer architecture (Vaswani et al., 2017) for tabular data and learns a feature graph that focuses on learning meaningful interaction between different features. DRSA-Net (Zheng et al., 2023) uses dual-route structure GNNs to learn adaptively the sparse graph structure. INCE (Villaizán-Vallelado et al., 2024) has a similar approach as T2G-Former, but uses an Interaction Network (Battaglia et al., 2016) instead of a Transformer. MPCFIN (Ye et al., 2024) uses cross-feature embeddings and multiplex GNNs to model the interaction and dependencies between features.

The feature GNN literature (Li et al., 2019a; Yan et al., 2023; Zheng et al., 2023; Villaizán-Vallelado et al., 2024; Ye et al., 2024; Zhou et al., 2022) suggests that the learned adjacency matrix can be used to interpret and explain the feature interactions. However, there are two reasons to be careful with this interpretation.

1. The evaluation of the graph structure is only heuristic to the best of our knowledge. The aforementioned explicit GTDL methods (FiGNN, INCE, T2G-Former, MPCFIN, and DRSA-Net) report the learned adjacency matrix for one or a few real-world datasets. They argue that the learned feature interactions are meaningful by post-hoc explaining the feature interactions. They justify the connections by referencing the semantic meaning of the feature names, suggesting that connected features are intuitively related. The issue with this approach is that, in the absence of a ground-truth graph structure, it becomes impossible to quantitatively evaluate the learned adjacency matrix.
2. The GTDL methods do not explicitly instruct the model to learn the true underlying graph structure. The loss is computed exclusively on the error between predicted and true target values. This means the model is only incentivized to improve predictive accuracy, not to accurately model the underlying feature interactions. As a result, the learned graph structure may not reliably reflect the true relationships between features. This limits its usefulness for interpretability and potentially constrains predictive performance.

Table2Graph (Zhou et al., 2022) addresses the first problem, that real-world datasets do not have a ground truth graph structure, by using a synthetic dataset. However, the learned graph structure is still evaluated heuristically against the ground-truth interactions, by visually comparing the learned weighted adjacency matrix with the ground-truth interactions. The second problem, that the model is only prediction-centric, is addressed by introducing a reinforcement learning term to the loss function to explore the adjacency matrix. This encourages the model to also focus on learning the graph structure, rather than just the predictive performance.

Not all GNN methods treat predicting the target feature in the same way. Some of these models treat predicting the target feature as a node-level task, while others treat it as a graph-level task (Prince, 2023). Node-level approaches (T2G-Former, INCE, MPCFIN) include a target node in the graph structure, and pass the embedding of the target node to an output layer. With this approach, the model learns a weighted adjacency matrix of size $p \times p$. Graph-level approaches (FiGNN, DSRA-Net, Table2Graph) do not include a target node, resulting in an adjacency matrix of size $(p-1) \times (p-1)$. The embeddings of all nodes are aggregated and passed to an output layer.

2.3 Probabilistic graphical model as a baseline for GTDL

Feature graphs, as studied by GTDL methods, share similarities with PGMs (Lauritzen, 1996; Koller & Friedman, 2009). PGMs provide a principled framework for modeling multivariate dependencies by encoding conditional independence relationships among random variables using graphs. Widely applied in Bayesian statistics, PGMs represent the structure of a probability distribution, often Gaussian, through a compact graph encoding the conditional independencies among variables. Bayesian techniques in PGMs, like BDgraph (Mohammadi & Wit, 2019), have demonstrated strong empirical performance in recovering interaction structures (Vogels et al., 2024). The ability to quantify uncertainty in the learned graph structure makes them a useful sanity check for GTDL methods.

2.4 Related approaches

There are other related approaches that are after closer inspection not relevant to our discussion on GTDL. In the literature on recommender systems and click-through rate, there has been a longer interest in a different notion of feature interactions, that of *cross features*; e.g., (Cheng et al., 2016; Guo et al., 2017; Lian et al., 2018; Wang et al., 2017; Cai et al., 2021; Wang et al., 2020; Song et al., 2019). These models focus on learning multiple weighted products of features to improve the prediction of the target feature. As noted by (Li et al., 2019a), this limits the capability to model interactions across different features flexibly and explicitly. We are interested in how to model the feature interactions explicitly on a graph. Therefore, we do not discuss these methods in further detail.

Tabular foundation models, TabPFN (Hollmann et al., 2025), TabICL (Qu et al., 2025) and LimiX (Zhang et al., 2025) have recently gained attention due to their high predictive performance on tabular data. Two key aspects of these models are their alternating instance-wise and feature-wise attention layers, and its ensembling predictions over multiple feature permutations. The feature-wise attention layers work similar as the attention-based methods discussed in Subsection 2.1, and therefore could have been interpreted as implicit GTDL methods. However, these tabular foundation models contain techniques that prevent a straightforward interpretation of the feature-wise attention map as a weighted adjacency matrix. TabPFN and LimiX encode groups of features collectively rather than individually. TabICL incorporates rotary positional embedding (RoPE) (Su et al., 2024) independent of the feature permutation, which alters the attention map. Therefore, we do not consider these models in this work.

Tree-based models (e.g., XGBoost (Chen & Guestrin, 2016), LightGBM (Ke et al., 2017) and CatBoost (Prokhorenkova et al., 2019)) remain a popular choice for tabular data. Nevertheless, these approaches do not explicitly represent feature interactions in a graphical format. Due to the nature of tree architectures, the learned feature interactions can not easily be extracted from the model.

3 Evaluating feature interactions in GTDL

The development of GTDL is hindered by the fact that the learned graph structure is only evaluated heuristically. To solve this, we introduce a framework to evaluate the learned graph structure of GTDL methods with synthetic datasets and quantitative metrics.

3.1 Synthetic data with a graph structure

Most existing GTDL methods lack rigorous evaluation of the learned graph structure. Typically, the learned graph structure is evaluated heuristically, by reporting a visualization of the learned weighted adjacency matrix of real-world datasets. Feature interactions are post-hoc explained based on the semantic meaning of the feature names. Evaluating only on real-world datasets is problematic, as the *true* graph structure is not known. Therefore, we propose using *synthetic* datasets. Using synthetic data enables GTDL methods to compare the learned graph structure with the ground-truth underlying graph structure in a controlled environment.

We adapt two existing data generation methods from the literature. The three-step process is sketched in Figure 2, and the details are given in Appendix B.

- **Multivariate normals (MVNs)** are typically studied by PGM methods. We follow the default procedure of generating conditional multivariate data (as described in (Mohammadi & Wit, 2015), for instance). In short, we employ the following: (i) Sample a graph structure $G_{\text{true}} \in \mathbb{R}^{p \times p}$ from the Bernoulli distribution. (ii) Sample a covariance matrix $\Sigma_G \in \mathbb{R}^{p \times p}$ from the G-Wishart distribution (Roverato, 2002; Letac & Massam, 2007) to describe the feature interactions. (iii) Obtain n samples $D \in \mathbb{R}^{n \times p}$ from $\mathcal{N}(0, \Sigma_G)$.
- **Structural causal model (SCM)** (Pearl, 2021) are used to generate tabular data in tabular foundation models (Hollmann et al., 2025; Qu et al., 2025; Zhang et al., 2025). We follow a simplified version of the data generation process in these tabular foundation models, that is: (i) Generate a directed acyclic graph (DAG) to define the graph structure of the SCM. To obtain the undirected graph G_{true} from the DAG, we moralize (connect all parents of a child node and drop the direction of the edges) and marginalize (drop the root nodes and connect its children) the DAG (Cowell et al., 1999). More details on moralization and marginalization are discussed in Appendix B; (ii) Sample computational maps f_i for each child node i in the DAG. The computational maps $\{f_i\}$ are smooth nonlinear functions that take all the values of the incoming edges as input, and the output is the value of the child node i; (iii) Traverse random data x_{roots} in topological order through the DAG to obtain n samples $D \in \mathbb{R}^{n \times p}$.

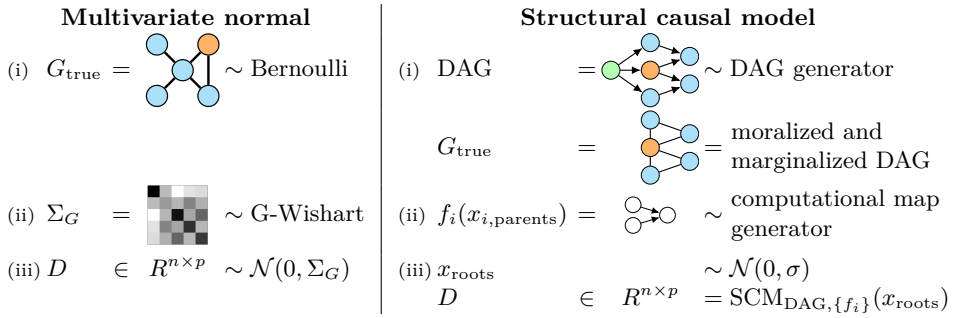

Figure 2: Two synthetic data generation pipelines. Both pipelines can roughly be divided into three steps. (i) Sample a graph structure; (ii) Sample feature interactions; (iii) Sample data given the graph and feature interactions. Nodes are colored as • cyan input features x, • orange target feature y, and • green root nodes x_{roots}.

For both approaches, we randomly select a target feature $y \in \mathbb{R}^{n \times 1}$ from D, with the remaining columns serving as input features $x \in \mathbb{R}^{n \times (p-1)}$. The target feature is directly influenced by its neighbors and only indirectly by non-neighbors. This setup lends itself well for evaluating the graph structure learned by GTDL methods. Consider the simple example: $x_0 - x_1 - x_2$. The model could learn to use x_0 to predict x_2 directly,

by learning that there is an edge between them. However, this is suboptimal, as x_2 is only indirectly related to x_0. The model should instead recognize x_1 as a better predictor for x_2, which results in learning the correct graph structure. Furthermore, the model should also learn that there is an edge in both directions, as the child of a parent can also be used to predict the parent.

The proposed synthetic data generation methods are different from the synthetic datasets used by TabNet (Arik & Pfister, 2020) or Table2Graph (Zhou et al., 2022), previously discussed in Subsections 2.1 and 2.2. All input features in these datasets are conditionally independent of each other, and have a direct interaction with the target feature. Therefore, there is no underlying graph structure to be learned, as all features are connected only to the target feature. In contrast, the MVN and SCM data generation methods create datasets with more complex underlying graph structures, making them more suitable to evaluate GTDL methods.

We acknowledge that our data generation processes may not be fully representative of real-world tabular data. For instance, the graphs evaluated in this work (Subsection 3.4 and Section 4) are relatively small ($p = 10$), the MVN does only have linear feature interactions, and the SCM does not have missing nodes within the DAG. However, key is that the datasets have a clear underlying ground-truth graph structure that GTDL methods should be able to learn. If models can not model the feature interactions of these synthetic datasets well, it is unlikely to expect that, on larger ($p \gtrsim 100$) and real-world datasets, these models will learn meaningful feature interactions.

3.2 Metric for evaluating feature interactions

Current GTDL methods only report the predictive performance of the target feature, and do not evaluate the learned feature interactions quantitatively. We propose to evaluate the quality of the graph structure by comparing edge-wise (ignoring the diagonal) the true binary adjacency matrix A_{true} with the learned weighted adjacency matrix $A_{\text{pred}} = A$ with the receiver operating characteristic area under curve (ROC AUC) (Bradley, 1997). This metric reflects to what degree the feature interaction strengths of the true edges are higher than those of the true non-edges. Ranging from 0 to 1, a value of 0.5 equals a random guess. A high ROC AUC indicates that all true edges have higher feature interaction strengths than all true non-edges. The ROC AUC is a 'relative measure', meaning that it is not sensitive to the scale of the feature interaction strengths. This way, we are forgiving in the evaluation of the feature interactions, as we only measure if the model can distinguish between true edges and true non-edges, and not the absolute values of the feature interaction strengths.

3.3 Pruning the feature interactions

To understand the effect of learning the correct graph structure, we model the GTDL methods in two different settings, which are sketched in Figure 3. First, we train the GTDL with a fully connected graph. This is the default setting in GTDL methods, as the true graph in real-world datasets is not known. Second, we limit feature interactions to only those present in the synthetic data, effectively *pruning the graph to the true edges*. This means that the model is only allowed to learn feature interactions that are present in the true graph. Practically, this is done by masking the attention map or the graph structure within the network architecture. This is only possible if the true graph is known, which is the case for synthetic data. By comparing the results from the fully connected and the pruned graphs, we can see how much the GTDL methods benefit from using only the true edges.

3.4 Setup of the experiments

We conduct a standard deep learning experiment to evaluate existing GTDL methods. That is, we optimize the prediction of the target feature y given the input features x. This is done with the default GTDL setting of a fully connected graph, and with the pruned setting where the feature interactions are limited to the true edges only. We tune the hyperparameters of the model and the learning rate, we use cross validation, and optimize the mean squared error (MSE) with Adam (Kingma & Ba, 2017). Further details on the splitting,

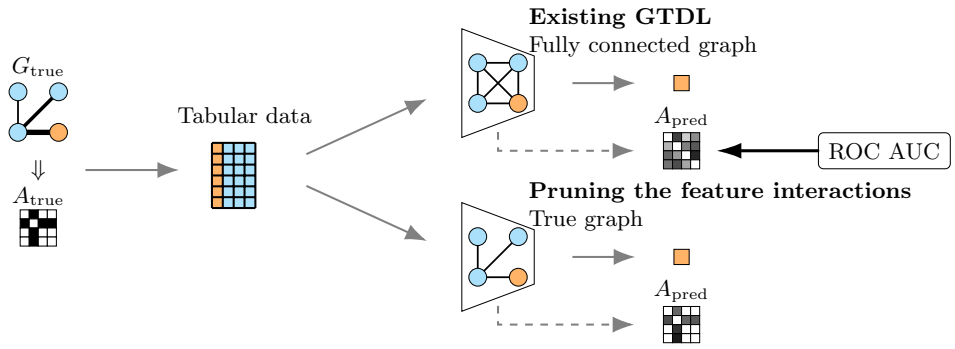

Figure 3: Upper branch (similar to Figure 1): Existing GTDL methods use a fully connected graph. The learned adjacency matrix A_{pred}, that we extract out of GTDL methods, is compared to the true adjacency matrix A_{true} with the ROC AUC to evaluate the learned feature interactions. Lower branch: When the graph is pruned to the true edges, GTDL methods can only model feature interactions that are present in the true graph. By doing this, we can analyze the effect when the correct graph structure is used in GTDL methods.

training, evaluation, and hyperparameter tuning and cross validations can be found in Appendix C. The code is publicly available.[2]

We report the quality of the learned graph structure (Subsection 3.2) only for the fully connected setting, as the pruned setting trivially has perfect graph quality. The pruned graph only contains the true edges, so the learned graph structure is equal to the true graph structure. For the predictive performance, we use the R2 score to compare the regression models. To average over the datasets, we use the *normalized* R2 score. This normalization is introduced by Wistuba et al. (2015) and used by Feurer et al. (2022); Grinsztajn et al. (2022) for instance. Per dataset, the R2 score is normalized between zero and one, using the worst-performing and the best-performing model on that dataset as the lower and upper bound, respectively.

We run the experiments for three datasets belonging to both dataset types (MVN and SCM), and their graph structures are shown in Appendix B. We compare all explicit GTDL methods that have publicly published code. That is, we compare FiGNN (Li et al., 2019a), T2G-Former (Yan et al., 2023) and INCE (Villaizán-Vallelado et al., 2024). The remainder of the explicit GTDL methods, DRSA-Net (Zheng et al., 2023), MPCFIN (Ye et al., 2024) and Table2Graph (Zhou et al., 2022), have not published their code repositories. For implicit, attention-based methods, we take FT-Transformer (Gorishniy et al., 2021) as an illustrative example. In Appendix A, we discuss how these methods use and interpret the learned weighted adjacency matrix, and how we adapt the implementations to compare them. We use the PGM method BDgraph (Mohammadi & Wit, 2019) as a baseline to understand how well GTDL methods should be able to learn the feature interactions.

4 Results of structure-aware learning in GTDL

To substantiate our claim that GTDL methods should focus on learning the graph structure, we demonstrate that GTDL methods do not accurately learn the feature interactions and that the predictive performance improves when the graph is pruned to its true edges. Results are aggregated per dataset type, see Appendix D for the results per dataset.

4.1 Feature interactions

The ROC AUC of the feature interactions is shown in Figure 4. For all GTDL methods, across both datasets, the ROC AUC is approximately 0.5, which is equal to random chance. There is no difference in the values of

[2] `https://anonymous.4open.science/r/gtdl_tmlr`

the adjacency matrix whether there exists a true edge or not. This shows that GTDL methods do not learn an accurate graph structure. Therefore, the learned feature interactions should not be used for interpretability or explainability. Increasing the number of training samples does not change the ROC AUC, indicating that the poor performance of the GTDL methods is not due to insufficient data.

PGMs, which focus on learning the graph structure, can learn the feature interactions, while GTDL methods cannot. The PGM method BDgraph has an ROC AUC very close to 1 for the MVN datasets. Even for the SCM datasets, which have nonlinear feature interactions, BDgraph can achieve reasonable ROC AUC values. The fact that this PGM method, a non-deep learning method, can learn the feature interactions, while these advanced GTDL methods cannot, suggests that GTDL methods have room for improvement in learning the graph structure.

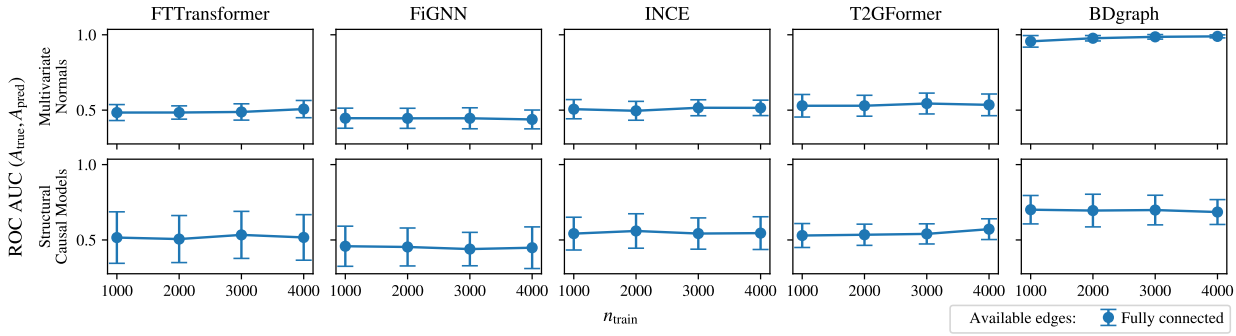

Figure 4: Graph quality in the form of the ROC AUC comparing the learned weighted adjacency matrix with the true binary one, for two different dataset types. All GTDL models have ROC AUC ≈ 0.5, which is random chance, indicating that they are not able to learn the feature interactions in any meaningful way. The PGM method BDgraph does learn the correct feature interactions better. Error bars show standard deviation across seeds, cross validations and datasets.

4.2 Predictive performance

The R2 score of the prediction of the target feature is shown in Figure 5. The key takeaway is that, in general, pruning the graph to the true edges improves the predictive performance of GTDL methods. This result indicates the importance of incorporating accurate structural information into GTDL models. When the graph is pruned to only include true edges, the models are less likely to overfit to spurious or irrelevant feature interactions. In contrast, fully-connected models must learn to ignore many false edges, which can introduce noise and make optimization more difficult, especially when data is limited. By restricting the model to only the true interactions, the learning process becomes more efficient and focused, leading to better generalization and higher predictive accuracy. This finding suggests that the inability of current GTDL methods to recover the true graph structure (as shown by the ROC AUC results) is not just a theoretical issue, but has practical consequences for predictive performance. If the true graph is known or can be estimated reliably, enforcing this structure can provide a boost in performance. We see the improvement of performance when pruning the graph for all evaluated GTDL methods except for FiGNN. In Appendix D.2, we discuss results that indicate that this is because FiGNN treats the task of predicting the target feature as a graph-level task, while all other models treat it as a node-level task.

The PGM method BDgraph performs, as expected, well on the MVN datasets, and poorly on the SCM datasets. The SCM datasets have nonlinear feature interactions, while BDgraph can only model linear feature interactions. This underlines the need for GTDL methods due to their flexibility to learn nonlinear relationships.

Furthermore, the benefit of incorporating the true graph increases by reducing the number of training samples. When ample data is available, models benefit less from incorporating the graph structure correctly, but when data is scarce, leveraging the graph structure improves the predictions. This is in line with the general notion of geometric deep learning, where symmetries in the data are used to improve the learning process (Bronstein

et al., 2021). When data is scarce, symmetries in the data are useful. However, when data is abundant, the model does not have to rely on symmetries in the data.

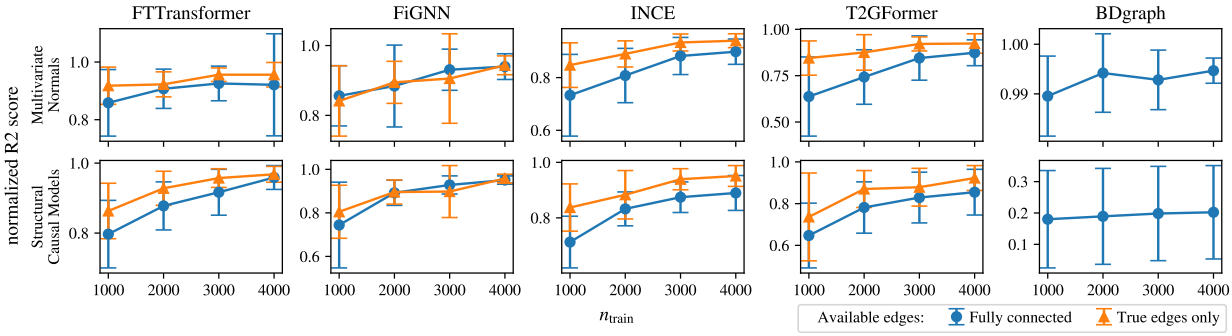

Figure 5: Predictive performance. When the graph is pruned to its true edges, the predictive performance is, in most cases, better compared to the fully connected graph. This difference reduces as the number of training samples increases. Note the different scale for the y-axis. Error bars show standard deviation across seeds, cross validations and datasets.

5 Conclusion, future work

In this work, we have analyzed the capability of graph-based tabular deep learning (GTDL) methods to learn feature interactions in tabular data. Inspired by the principles of probabilistic graphical models (PGMs), we proposed to use synthetic tabular datasets with known ground-truth graph structures, enabling the GTDL community to quantitatively assess whether models accurately capture the intended graph structure. Current GTDL approaches often produce graph structures used for interpretation, yet our analysis shows that these structures fail to reflect the true interactions among features. This indicates that the mechanisms of message-passing in GNNs, and attention in transformers, does not work as intended for tabular data. Our empirical findings demonstrate that when models operate on accurate interaction structures, predictive performance improves, highlighting that structural fidelity is not merely a matter of explainability, but a core driver of performance.

We highlight three directions how the analysis of this work can be extended in future work. First, future work should move beyond evaluating the learned graph structure (i.e., the presence of edges), but also consider the functional form of the feature interactions (i.e., the type of edges). Learning *how* features interact, and *in what way*, allows for more nuanced, robust, and interpretable modeling of feature relationships. Second, the evaluated datasets and graphs could be more expressive and challenging. Examples include larger graphs with richer topology, missing nodes, and more complex feature interactions involving categorical features, as well as real-world datasets with known ground-truth structure (e.g., knowledge graphs). However, our results showed that GTDL methods struggle to learn relatively simple, small graph structures, which suggests that improving robustness and structure induction on basic cases remains a priority before scaling to such settings. Finally, structure-aware modeling should be extended beyond flat tables to richer data modalities. Time-series data (Padhi et al., 2021) and relational databases (Fey et al., 2023; Robinson et al., 2024; Cong et al., 2024; Dwivedi et al., 2025) present new challenges for learning and validating feature interactions over time or across relational contexts. Relational deep learning (RDL) (Fey et al., 2023) could ask the same question as we propose GTDL should do: how do columns/features within a table relate? Currently, table-row embeddings in RDL lack structural bias from the graph structure of columns. Furthermore, our approach could be extended from table-level to relational database-level. When doing node-level prediction, does RDL rely on meaningful primary-foreign relationships?

Future work should build on the insights of this work to develop GTDL methods that more effectively learn and leverage feature interactions in tabular data. By prioritizing structural fidelity alongside predictive accuracy, future GTDL models can unlock their full potential.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 2623–2631, Anchorage AK USA, July 2019. ACM. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330701. URL https://dl.acm.org/doi/10.1145/3292500.3330701.

Sercan O. Arik and Tomas Pfister. TabNet: Attentive Interpretable Tabular Learning, December 2020. URL http://arxiv.org/abs/1908.07442.

Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/3147da8ab4a0437c15ef51a5cc7f2dc4-Paper.pdf.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, October 2018. URL http://arxiv.org/abs/1806.01261.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. In *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://papers.nips.cc/paper_files/paper/2011/hash/86e8f7ab32cfd12577bc2619bc635690-Abstract.html.

Andrew P. Bradley. The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, 30(7):1145–1159, July 1997. ISSN 00313203. doi: 10.1016/S0031-3203(96)00142-2. URL https://linkinghub.elsevier.com/retrieve/pii/S0031320396001422.

Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges, May 2021. URL http://arxiv.org/abs/2104.13478.

Shaofeng Cai, Kaiping Zheng, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Meihui Zhang. ARM-Net: Adaptive Relation Modeling Network for Structured Data. In *Proceedings of the 2021 International Conference on Management of Data*, pp. 207–220, Virtual Event China, June 2021. ACM. ISBN 978-1-4503-8343-1. doi: 10.1145/3448016.3457321. URL https://dl.acm.org/doi/10.1145/3448016.3457321.

Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, August 2016. doi: 10.1145/2939672.2939785. URL http://arxiv.org/abs/1603.02754.

Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil, Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, and Hemal Shah. Wide & Deep Learning for Recommender Systems, June 2016. URL http://arxiv.org/abs/1606.07792.

Tianji Cong, Madelon Hulsebos, Zhenjie Sun, Paul Groth, and H. V. Jagadish. Observatory: Characterizing Embeddings of Relational Tables, January 2024. URL http://arxiv.org/abs/2310.07736.

Robert G. Cowell, Steffen L. Lauritzen, A. Philip David, David J. Spiegelhalter, V. Nair, J. Lawless, and M. Jordan. *Probabilistic Networks and Expert Systems*. Springer-Verlag, Berlin, Heidelberg, 1 edition, 1999. ISBN 0-387-98767-3.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, May 2019. URL http://arxiv.org/abs/1810.04805.

Vijay Prakash Dwivedi, Sri Jaladi, Yangyi Shen, Federico López, Charilaos I. Kanatsoulis, Rishi Puri, Matthias Fey, and Jure Leskovec. Relational Graph Transformer, May 2025. URL http://arxiv.org/abs/2505.10960.

Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning. *Journal of Machine Learning Research*, 23(261):1–61, 2022. ISSN 1533-7928. URL http://jmlr.org/papers/v23/21-0992.html.

Matthias Fey, Weihua Hu, Kexin Huang, Jan Eric Lenssen, Rishabh Ranjan, Joshua Robinson, Rex Ying, Jiaxuan You, and Jure Leskovec. Relational Deep Learning: Graph Representation Learning on Relational Databases, December 2023. URL http://arxiv.org/abs/2312.04615.

Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, April 1980. ISSN 0340-1200, 1432-0770. doi: 10.1007/BF00344251. URL http://link.springer.com/10.1007/BF00344251.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting Deep Learning Models for Tabular Data. In *Advances in Neural Information Processing Systems*, volume 34, pp. 18932–18943. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/9d86d83f925f2149e9edb0ac3b49229c-Abstract.html.

Anirudh Goyal and Yoshua Bengio. Inductive Biases for Deep Learning of Higher-Level Cognition, August 2022. URL http://arxiv.org/abs/2011.15091.

Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data?, July 2022. URL http://arxiv.org/abs/2207.08815.

Huifeng Guo, Ruiming Tang, Yunming Ye, Zhenguo Li, and Xiuqiang He. DeepFM: A Factorization-Machine based Neural Network for CTR Prediction, March 2017. URL http://arxiv.org/abs/1703.04247.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, January 2025. ISSN 0028-0836, 1476-4687. doi: 10.1038/s41586-024-08328-6. URL https://www.nature.com/articles/s41586-024-08328-6.

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. TabTransformer: Tabular Data Modeling Using Contextual Embeddings, December 2020. URL http://arxiv.org/abs/2012.06678.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html.

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. URL http://arxiv.org/abs/1412.6980.

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, 2009. ISBN 978-0-262-25835-7.

Jannik Kossen, Neil Band, Clare Lyle, Aidan N Gomez, Thomas Rainforth, and Yarin Gal. Self-Attention Between Datapoints: Going Beyond Individual Input-Output Pairs in Deep Learning. In *Advances in Neural Information Processing Systems*, volume 34, pp. 28742–28756. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/f1507aba9fc82ffa7cc7373c58f8a613-Abstract.html.

Steffen L. Lauritzen. *Graphical Models*. Number 17 in Oxford Statistical Science Series. Clarendon Press ; Oxford University Press, Oxford : New York, 1996. ISBN 978-0-19-852219-5.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4):541–551, December 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.4.541. URL https://ieeexplore.ieee.org/abstract/document/6795724.

Gérard Letac and Hélène Massam. Wishart distributions for decomposable graphs. *The Annals of Statistics*, 35(3):1278–1323, July 2007. ISSN 0090-5364, 2168-8966. doi: 10.1214/009053606000001235. URL https://projecteuclid.org/journals/annals-of-statistics/volume-35/issue-3/Wishart-distributions-for-decomposable-graphs/10.1214/009053606000001235.full.

Cheng-Te Li, Yu-Che Tsai, Chih-Yao Chen, and Jay Chiehen Liao. Graph Neural Networks for Tabular Data Learning: A Survey with Taxonomy and Directions, January 2024. URL http://arxiv.org/abs/2401.02143.

Zekun Li, Zeyu Cui, Shu Wu, Xiaoyu Zhang, and Liang Wang. Fi-GNN: Modeling feature interactions via graph neural networks for CTR prediction. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, Cikm '19, pp. 539–548, Beijing, China and New York, NY, USA, 2019a. Association for Computing Machinery. ISBN 978-1-4503-6976-3. doi: 10.1145/3357384.3357951. URL https://doi.org/10.1145/3357384.3357951.

Zekun Li, Zeyu Cui, Shu Wu, Xiaoyu Zhang, and Liang Wang. Fi-GNN: Modeling Feature Interactions via Graph Neural Networks for CTR Prediction, October 2019b. URL https://arxiv.org/abs/1910.05552v1.

Zekun Li, Zeyu Cui, Shu Wu, Xiaoyu Zhang, and Liang Wang. Fi-GNN: Modeling Feature Interactions via Graph Neural Networks for CTR Prediction, July 2020. URL http://arxiv.org/abs/1910.05552.

Jianxun Lian, Xiaohuan Zhou, Fuzheng Zhang, Zhongxia Chen, Xing Xie, and Guangzhong Sun. xDeepFM: Combining Explicit and Implicit Feature Interactions for Recommender Systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1754–1763, London United Kingdom, July 2018. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3220023. URL https://dl.acm.org/doi/10.1145/3219819.3220023.

Duncan McElfresh, Sujay Khandagale, Jonathan Valverde, Vishak Prasad C, Benjamin Feuer, Chinmay Hegde, Ganesh Ramakrishnan, Micah Goldblum, and Colin White. When Do Neural Nets Outperform Boosted Trees on Tabular Data?, July 2024. URL http://arxiv.org/abs/2305.02997.

A. Mohammadi and E. C. Wit. Bayesian Structure Learning in Sparse Gaussian Graphical Models. *Bayesian Analysis*, 10(1), March 2015. ISSN 1936-0975. doi: 10.1214/14-BA889. URL http://arxiv.org/abs/1210.5371.

Reza Mohammadi and Ernst C. Wit. BDgraph: An R Package for Bayesian Structure Learning in Graphical Models. *Journal of Statistical Software*, 89:1–30, May 2019. ISSN 1548-7660. doi: 10.18637/jss.v089.i03. URL https://doi.org/10.18637/jss.v089.i03.

Inkit Padhi, Yair Schiff, Igor Melnyk, Mattia Rigotti, Youssef Mroueh, Pierre Dognin, Jerret Ross, Ravi Nair, and Erik Altman. Tabular Transformers for Modeling Multivariate Time Series. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3565–3569, June 2021. doi: 10.1109/ICASSP39728.2021.9414142. URL https://ieeexplore.ieee.org/abstract/document/9414142.

Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, Cambridge, second edition, reprinted with corrections 2021 edition, 2021. ISBN 978-0-511-80316-1.

Simon J.D. Prince. *Understanding Deep Learning*. The MIT Press, 2023. URL http://udlbook.com.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. CatBoost: Unbiased boosting with categorical features, January 2019. URL http://arxiv.org/abs/1706.09516.

Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A Tabular Foundation Model for In-Context Learning on Large Data, May 2025. URL http://arxiv.org/abs/2502.05564.

Joshua Robinson, Rishabh Ranjan, Weihua Hu, Kexin Huang, Jiaqi Han, Alejandro Dobles, Matthias Fey, Jan E. Lenssen, Yiwen Yuan, Zecheng Zhang, Xinwei He, and Jure Leskovec. RelBench: A benchmark for deep learning on relational databases. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 21330–21341. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/25cd345233c65fac1fec0ce61d0f7836-Paper-Datasets_and_Benchmarks_Track.pdf.

David Wilson Romero Guzman. *The Good, the Efficient and the Inductive Biases:: Exploring Efficiency in Deep Learning Through the Use of Inductive Biases*. PhD-Thesis - Research and graduation internal, Vrije Universiteit Amsterdam, September 2024.

Alberto Roverato. Hyper Inverse Wishart Distribution for Non-decomposable Graphs and its Application to Bayesian Inference for Gaussian Graphical Models. *Scandinavian Journal of Statistics*, 29(3):391–411, 2002. ISSN 1467-9469. doi: 10.1111/1467-9469.00297. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-9469.00297.

Gowthami Somepalli, Micah Goldblum, Avi Schwarzschild, C. Bayan Bruss, and Tom Goldstein. SAINT: Improved Neural Networks for Tabular Data via Row Attention and Contrastive Pre-Training, June 2021. URL http://arxiv.org/abs/2106.01342.

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks, August 2019. URL http://arxiv.org/abs/1810.11921.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. RoFormer: Enhanced transformer with Rotary Position Embedding. *Neurocomputing*, 568:127063, February 2024. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.127063. URL https://www.sciencedirect.com/science/article/pii/S0925231223011864.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic Attribution for Deep Networks, June 2017. URL http://arxiv.org/abs/1703.01365.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017. URL http://arxiv.org/abs/1706.03762.

Jesse Vig. A Multiscale Visualization of Attention in the Transformer Model. In Marta R. Costa-jussà and Enrique Alfonseca (eds.), *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 37–42, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-3007. URL https://aclanthology.org/P19-3007/.

Mario Villaizán-Vallelado, Matteo Salvatori, Belén Carro, and Antonio Javier Sanchez-Esguevillas. Graph Neural Network contextual embedding for Deep Learning on tabular data. *Neural Networks*, 173:106180, May 2024. ISSN 0893-6080. doi: 10.1016/j.neunet.2024.106180. URL https://www.sciencedirect.com/science/article/pii/S0893608024001047.

Lucas Vogels, Reza Mohammadi, Marit Schoonhoven, and Ş. İlker Birbil. Bayesian Structure Learning in Undirected Gaussian Graphical Models: Literature Review with Empirical Comparison. *Journal of the American Statistical Association*, 119(548):3164–3182, October 2024. ISSN 0162-1459. doi: 10.1080/01621459.2024.2395504. URL https://doi.org/10.1080/01621459.2024.2395504.

Ruoxi Wang, Bin Fu, Gang Fu, and Mingliang Wang. Deep & Cross Network for Ad Click Predictions. In *Proceedings of the ADKDD'17*, pp. 1–7, Halifax NS Canada, August 2017. ACM. ISBN 978-1-4503-5194-2. doi: 10.1145/3124749.3124754. URL https://dl.acm.org/doi/10.1145/3124749.3124754.

Ruoxi Wang, Rakesh Shivanna, Derek Z. Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed H. Chi. DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems, October 2020. URL http://arxiv.org/abs/2008.13535.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. Learning hyperparameter optimization initializations. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pp. 1–10, October 2015. doi: 10.1109/DSAA.2015.7344817. URL https://ieeexplore.ieee.org/abstract/document/7344817.

Jiahuan Yan, Jintai Chen, Yixuan Wu, Danny Z. Chen, and Jian Wu. T2G-Former: Organizing tabular features into relation graphs promotes heterogeneous feature interaction. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, volume 37 of *AAAI'23/IAAI'23/EAAI'23*, pp. 10720–10728. AAAI Press, February 2023. ISBN 978-1-57735-880-0. doi: 10.1609/aaai.v37i9.26272. URL https://doi.org/10.1609/aaai.v37i9.26272.

Mang Ye, Yi Yu, Ziqin Shen, Wei Yu, and Qingyan Zeng. Cross-Feature Interactive Tabular Data Modeling With Multiplex Graph Neural Networks. *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–15, 2024. ISSN 1558-2191. doi: 10.1109/TKDE.2024.3440654. URL https://ieeexplore.ieee.org/document/10631296/?arnumber=10631296.

Xingxuan Zhang, Gang Ren, Han Yu, Hao Yuan, Hui Wang, Jiansheng Li, Jiayun Wu, Lang Mo, Li Mao, Mingchao Hao, Ningbo Dai, Renzhe Xu, Shuyang Li, Tianyang Zhang, Yue He, Yuanrui Wang, Yunjia Zhang, Zijing Xu, Dongzhe Li, Fang Gao, Hao Zou, Jiandong Liu, Jiashuo Liu, Jiawei Xu, Kaijie Cheng, Kehan Li, Linjun Zhou, Qing Li, Shaohua Fan, Xiaoyu Lin, Xinyan Han, Xuanyue Li, Yan Lu, Yuan Xue, Yuanyuan Jiang, Zimu Wang, Zhenlei Wang, and Peng Cui. LimiX: Unleashing Structured-Data Modeling Capability for Generalist Intelligence, September 2025. URL http://arxiv.org/abs/2509.03505.

Qinghua Zheng, Zhen Peng, Zhuohang Dang, Linchao Zhu, Ziqi Liu, Zhiqiang Zhang, and Jun Zhou. Deep Tabular Data Modeling With Dual-Route Structure-Adaptive Graph Networks. *IEEE Transactions on Knowledge and Data Engineering*, 35(9):9715–9727, September 2023. ISSN 1558-2191. doi: 10.1109/TKDE.2023.3249186. URL https://ieeexplore.ieee.org/document/10054100.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph Neural Networks: A Review of Methods and Applications, October 2021. URL http://arxiv.org/abs/1812.08434.

Kaixiong Zhou, Zirui Liu, Rui Chen, Li Li, Soo-Hyun Choi, and Xia Hu. Table2Graph: Transforming Tabular Data to Unified Weighted Graph. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence*, pp. 2420–2426, Vienna, Austria, July 2022. International Joint Conferences on Artificial Intelligence Organization. ISBN 978-1-956792-00-3. doi: 10.24963/ijcai.2022/336. URL https://www.ijcai.org/proceedings/2022/336.

A Implementations from literature

This section contains implementation details of the GTDL models that are evaluated in Section 4. In the following Appendices A.1 to A.4, each first paragraph explains what kind of graph is learned, and where and how this is used by the original authors. In the subsections, we discuss the adaptations we made to make the learned graph structures compatible with our discussion and implementation.

Additional notation. We use N for the number of samples, L as the number of layers in the network, H as the number of transformer heads, and p as the number of features. To indicate values that are ranging between 0 and 1, we denote the corresponding set as $\mathbb{R}_{[0,1]}$.

Interpreting the attention map as a weighted adjacency matrix. For most methods (FT-Transformer, T2G-Former and FiGNN), the learned graph comes from averaging the attention map $a \in \mathbb{R}_{[0,1]}^{N \times L \times H \times p \times p}$ over the samples, layers, and heads. This attention map is normalized with softmax across the last dimension. We note individual values from the attention map as a_{ilhjk}, where i is the sample index, l is the layer index, h is the head index, and j, k are the feature indices. So the average attention map is $a_{jk} = \frac{1}{N \times L \times H} \sum_{ilh} a_{ilhjk} \in \mathbb{R}_{[0,1]}^{p \times p}$.

We want to interpret the average attention map a_{jk} as the weighted adjacency matrix A_{jk}. For this, we have to 'denormalize' the average attention map. As the attention map a is normalized with a softmax, the last dimension (the rows in the attention map per individual layer and head) sum to 1, such that $\sum_k a_{ilhjk} = \mathbb{1}_{ilhj}$. This gives a problem, as the maximum value the attention map a_{ilhjk} can have cannot have two values close to 1 in the same row, while the weighted adjacency matrix A_{jk} should be able to have multiple values close to 1 in the same row.

To 'denormalize' the attention map, we add two steps. First, we set the diagonal of the attention map to zero, as the self-interactions should not be taken into account during evaluation of the feature interactions:

$$a_{ilhjk} = 0 \quad \forall \quad j = k.$$

Second, we divide the attention maps by the maximum value across the row to obtain the adjacency matrices:

$$A_{ilhjk} = a_{ilhjk} / \max_k(a_{ilhjk}).$$

By doing this, all the values with the highest attention across that row now have a value of 1 in the weighted adjacency matrix. Ignoring the diagonal in the attention map is a key step in this procedure: If the model learns that it should not give high attention to the non-diagonal values (as those features are not related), the model should learn to give high attention to the diagonal values. The diagonal values are excluded in the denormalization and do not affect the adjacency matrix.

A.1 FT-Transformer

FT-Transformer (Gorishniy et al., 2021) (`https://github.com/yandex-research/rtdl-revisiting-models`) learns the attention map $a \in \mathbb{R}_{[0,1]}^{N \times L \times H \times p \times p}$. (Gorishniy et al., 2021) interpret in Section 5.3 the average attention map $a_{jk} = \frac{1}{N \times L \times H} \sum_{ilh} a_{ilhjk} \in \mathbb{R}_{[0,1]}^{p \times p}$ as feature importance. For a few real-world datasets, they compare it to Integrated Gradients (Sundararajan et al., 2017) using rank correlation and find that it performs similarly.

As the attention map is normalized with the softmax, we denormalize it to obtain the weighted adjacency matrix A as described above. This is the only post-hoc adaptation we made to the original implementation.

A.2 T2G-Former

T2G-Former (Yan et al., 2023) (`https://github.com/jyansir/t2g-former`) learn a feature-relation graph (FR-Graph) $a \in \mathbb{R}_{[0,1]}^{N \times L \times H \times p \times p}$ (Equation 6 in (Yan et al., 2023)). The strength of the graph can be interpreted as the strength of the relations between the features. Section 5.3 and Figure 3 in (Yan et al., 2023) show the FR-Graph for two real-world datasets.

Instead of the Hadamard product in equation 6 in (Yan et al., 2023) to construct the FR-Graph, we use a sum, consistent with their code implementation of the FR-Graph. (Yan et al., 2023) present the FR-Graph from the first and the last layer of the network. We assume that these are averaged over the samples and the heads. Instead, we average over all the layers, following the approach of FT-Transformer. As the FR-Graph is normalized with the softmax, we denormalize the FR-Graph to obtain the weighted adjacency matrix A as described above.

A.3 INCE

INCE (Villaizán-Vallelado et al., 2024) (`https://github.com/MatteoSalvatori/INCE`) learn edge embeddings $e \in \mathbb{R}^{N \times (p(p-1)) \times d_{\text{emb}}}$ with d_{emb} the embedding dimension and $(p(p-1))$ the number of edges in

a fully connected graph excluding self-loops. Section 6.2 of (Villaizán-Vallelado et al., 2024) presents an algorithm to calculate the feature-feature interaction $p_{\text{int}} \in \mathbb{R}_{[0,1]}^{p \times p}$ from the edge embeddings. Figure 11 in (Villaizán-Vallelado et al., 2024) shows the feature-feature interaction p_{int} on a real-world dataset.

A lower value of p_{int} implies more significance. Therefore, we apply one additional step to obtain the weighted adjacency matrix $A = 1 - p_{\text{int}}$.

A.4 FiGNN

FiGNN (Li et al., 2019a) (`https://github.com/CRIPAC-DIG/Fi_GNN/tree/7e207b2ffb4f25b63d2079cf7761d09e5dedf6e8`[3]) learn a feature graph in the form of attentional edge weights $a \in \mathbb{R}_{[0,1]}^{N \times (p-1) \times (p-1)}$ for the input features (equations 4 and 5 in Li et al. (2019a)). The edge weights are interpreted as the importance of the interactions. Therefore, they are used to providing explanations on the relationship between different features. In Section 4.5 and Figure 5, the edge weights are presented as a heat map, and are used to explain the relations between features on a real-world dataset.

As the attention edge weights are normalized with the softmax, we denormalize them to obtain the weighted adjacency matrix A as described above. The code implementation of FiGNN is published with TensorFlow. As our implementation is in PyTorch, we have adapted the code to PyTorch. The learned adjacency matrix is of size $(p-1) \times (p-1)$, we impute an additional row and column for the target feature with values of zero.

There are different implementation versions of FiGNN. The first version of FiGNN has been presented at CIKM in November 2019, which is identical to version 1 on ArXiv (Li et al., 2019b). We use the implementation of this version. In July 2020, version 2 on ArXiv (Li et al., 2020) was published, and the main repository (`https://github.com/CRIPAC-DIG/Fi_GNN/`) was updated accordingly. Version 2 has some additional attention layers. Furthermore, when inspecting the published code. We observed that this second version does not have a trainable feature graph in its code implementation. Therefore, we stick to the original code implementation of version 1 (`https://github.com/CRIPAC-DIG/Fi_GNN/tree/7e207b2ffb4f25b63d2079cf7761d09e5dedf6e8`).

B Data generation

In this section, we describe the graph and data generation process of the two synthetic dataset approaches introduced in Subsection 3.1 and their hyperparameters used in Section 4.

Multivariate normals. We follow the default procedure of generating conditional multivariate data, (Mohammadi & Wit, 2015):

(i) Sample a true graph structure $G_{\text{true}} \in \mathbb{R}^{p \times p}$ from the Bernoulli distribution with an edge inclusion probability P_{edge}.

(ii) Sample a covariance matrix $\Sigma_G \in \mathbb{R}^{p \times p}$ from the G-Wishart distribution (Roverato, 2002; Letac & Massam, 2007), which is conditioned on the graph structure G_{true};[4]

(iii) Obtain n samples $D \in \mathbb{R}^{n \times p}$ from a multivariate normal distribution $\mathcal{N}(0, \Sigma_G)$.

In our experiments, we have $p = 10$ nodes, and an edge inclusion probability of $P_{\text{edge}} = 0.267$. This results in the graph structures as depicted in Figure 6.

[3]This on purpose a specific commit, as there are different implementations.

[4]In fact, we sample the precision matrix $K_G = \Sigma_G^{-1}$ form the G-Wishart distribution, and invert it to obtain the covariance matrix Σ_G. The underscore $\cdot_G$ indicates that the matrix is conditioned on the graph structure G_{true}.

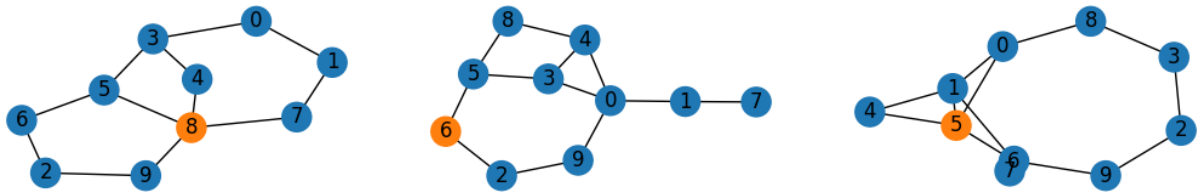

Figure 6: Graphs used in experiments for the MVN1, MVN2 and MVN3 datasets.

Structural causal models. We follow a similar setup as (Hollmann et al., 2025) to generate an SCM and sample data conditional on the graph. They show that with their setup, the synthesized data is similar to real-world tabular data.

(i) Randomly sample a DAG, with n_{root} root nodes and p child nodes with a probability of P_{edge} of an incoming edge. The undirected graph structure G_{true} is obtained by moralizing and marginalizing the DAG (Cowell et al., 1999). Moralizing makes the graph undirected by dropping the direction of the existing edges and connecting all parents of a child node. Marginalizing removes the root nodes from the graph, as they are not part of the dataset D. When marginalizing a node, we connect all neighbors of the removed node to each other. In this work, only the root nodes are marginalized. This makes the order of moralization and marginalization irrelevant. The red lines in Figure 7 show the new edges that are added by moralization and marginalization.

(ii) Randomly sample deterministic computational mappings f_i for each child node i in the graph, where the mappings are smooth nonlinear functions, randomly picked from the set of maps listed in Table 2. A computational map defines how a child node i is computed from its parents. They take all the values of the incoming edges as input, and the output is the value of the child node i.

(iii) Randomly sample root nodes $x_{\text{root}} \sim \mathcal{N}(0,1) \in \mathbb{R}^{n \times n_{\text{root}}}$ and traverse the DAG in a topological order, $x_i = f_i(x_{i,\text{parents}}) \in \mathbb{R}^{n \times 1}$. Each output x_i is normalized, clipped between $(-3, 3)$, and Gaussian noise $\mathcal{N}(0, 0.5)$ is added. This is summarized by

$$x_i = \text{clip}(\text{normalize}(f_i(x_{i,\text{parents}})) + \mathcal{N}(0, 0.5), -3, 3). \tag{1}$$

We consider all the traversed outputs x_i as the dataset $D \in \mathbb{R}^{n \times p}$.

Each DAG has $p = 10$ nodes. These p nodes are evenly distributed over $n_{\text{DAG layers}} = 3$ layers, where each layer has a minimum of 3 nodes. The DAG has a 'zeroth' layer of $n_{\text{root}} = 3$ root nodes. This means that each layer has 3 or 4 nodes. Each node has a $P_{\text{edge}} = 0.5$ probability of having an edge to the nodes in the next layer. With these hyperparameters, the three DAGs that are used in Section 4 are shown in Figure 7.

Table 2: Computational maps

# parents	$f(x_{\text{parents}})$
1	$x_1^2/3$
	$0{,}5\,x_1^2 + 3\,x_1$
	$-\lvert x_1\rvert + 4\,x_1$
2	$(x_1 x_2 + x_1^2)/2$
	$x_1^2 + x_2^2 - x_1 x_2$
	$-(x_1 + x_2)^2 + x_1 x_2$
3	$(x_1 x_2 + x_3^2)/3$
	$-x_1^2 + x_2 x_3 + x_3$
	$(x_1 + x_2 + x_3) + x_1 x_3$

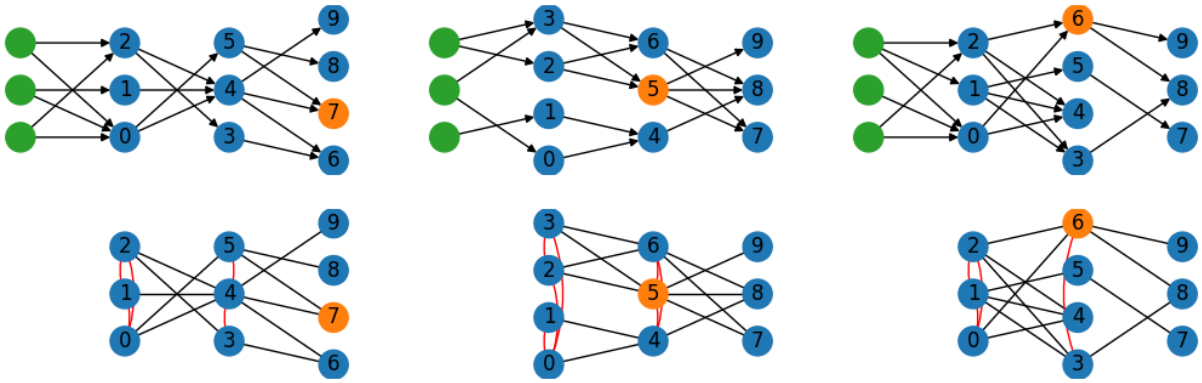

Figure 7: Top: DAGs used in experiments for the SCM1, SCM2 and SCM3 datasets. Bottom: Corresponding undirected graph structures G_{true} after moralization and marginalization. Moralized and marginalized edges are depicted in red.

C Experiment details

Data splitting. We adapt our train, validation, and test splitting and our tuning strategy to balance between a fair comparison between different dataset sizes and an efficient hyperparameter tuning.

Following (Grinsztajn et al., 2022), we differentiate between a validation set used for early stopping $D_{\text{val, early stop}}$ and a validation set used for hyperparameter tuning $D_{\text{val, hparam}}$, such that we have four disjoint sets: D_{train}, $D_{\text{val, early stop}}$, $D_{\text{val, hparam}}$, and D_{test}. We vary the number of training samples n_{train} in our experiments between 1000 and 4000, and set both $n_{\text{test}} = n_{\text{val, hparam}} = 2500$ and $n_{\text{val, early stop}} = 0.25n_{\text{train}}$.

In our experiments, we do not change $D_{\text{val, hparam}}$ and D_{test} to limit the number of cross-validation and iterations we have to do. We randomly sample D_{train} and $D_{\text{val, early stop}}$ for each fold. This strategy is visualized in Figure 8. For $n_{\text{train}} = 1000$ samples we evaluate over 4 folds, $n_{\text{train}} = 2000$ over 3 fold, for $n_{\text{train}} = 3000$ over 2 folds and for $n_{\text{train}} = 4000$ over 1 fold.

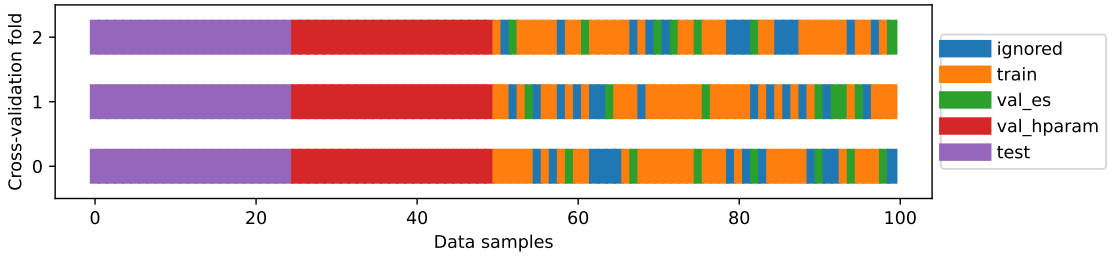

Figure 8: Splitting strategy for the example $N = 100$ and $n_{\text{train}} = 0.3N$. The training set and the validation set for early stopping are randomly sampled for each fold. The test set and validation set for hyperparameter tuning are fixed.

Training and evaluation. We minimize the MSE loss function and optimize using Adam (Kingma & Ba, 2017) with a fixed batch size of 256 and tune the learning rate together with the other model hyperparameters. We continue training until the validation loss does not improve for 10 epochs. There is a theoretical upper bound of 400 epochs, which is rarely reached in practice. We select the best hyperparameters that minimize the MSE on the separate hyperparameter validation set. After tuning, we run 10 runs per cross-validation fold. We report the R2 score to evaluate the predictive performance of the target feature, and the ROC AUC to evaluate the learned feature interactions.

Hyperparameter tuning. For every combination of network, dataset, and n_{train}, we tune the model's hyperparameters and the learning rate. For all models, we use tree-structured Parzen estimator (TPE) (Bergstra et al., 2011), a Bayesian optimization technique within the Optuna library (Akiba et al., 2019). We run a total of 50 trials for each setting, where the first trial has the default hyperparameters of the implementation. We keep the default setting of the Optuna implementation, where the first 10 trials are done with random search.

The hyperparameter distribution and the default hyperparameters of all models are listed in Tables 3 to 6. For all models, the search space and default values are taken from the original implementations if not specified otherwise in the caption of the tables. The search space of layer count and embedding size is set the same for fairer comparison across models. The distribution space of the learning rate is LogUniform$[10^{-5}, 10^{-3}]$ with a default value of 10^{-3} for all models.

Table 3: FT-Transformer (Gorishniy et al., 2021) hyperparameter space.

Parameter	Distribution	Default
Layer count	UniformInt$[1, 6]$	3
Embedding size	$\{8, 16, 32, 64, 128, 264\}$	128
Attention head count	-	8
Attention dropout	Uniform$[0.0, 0.5]$	0.2
FFN size factor	Uniform$[2/3, 7/3]$	$4/3$
FFN dropout	Uniform$[0.0, 0.5]$	0.1
Residual dropout	Uniform$[0.0, 0.2]$	0

Table 4: T2G-Former (Yan et al., 2023) hyperparameter space. Default values are taken from the same as from FT-Transformer.

Parameter	Distribution	Default
Layer count	UniformInt$[1, 6]$	3
Embedding size	$\{8, 16, 32, 64, 128, 264\}$	128
Attention head count	-	8
Attention dropout	Uniform$[0.0, 0.5]$	0.2
FFN size factor	Uniform$[2/3, 7/3]$	$4/3$
FFN dropout	Uniform$[0.0, 0.5]$	0.1

Table 5: INCE (Villaizán-Vallelado et al., 2024) hyperparameter space.

Parameter	Distribution	Default
Layer count	UniformInt$[1, 6]$	4
Embedding size	$\{8, 16, 32, 64, 128, 264\}$	128
MLP layer count	$\{1, 2, 3, 4\}$	3
Dropout	Uniform$[0.0, 0.5]$	0

Table 6: FiGNN (Li et al., 2019a) hyperparameter space. The distribution space was not shared by the original implementation.

Parameter	Distribution	Default
Layer count	UniformInt$[1, 6]$	3
Embedding size	$\{8, 16, 32, 64, 128, 264\}$	16
Dropout	Uniform$[0.0, 0.5]$	0

D Additional results

D.1 Results per dataset

In Section 4 we have discussed the results of the learned graph structure and the predictive performance of the target feature while aggregating over three datasets per the dataset type MVN and SCM. In Figure 9 and Figure 10, we show the results per individual dataset. The results are consistent with the results shown in Figure 4 and Figure 5; no new insights are gained.

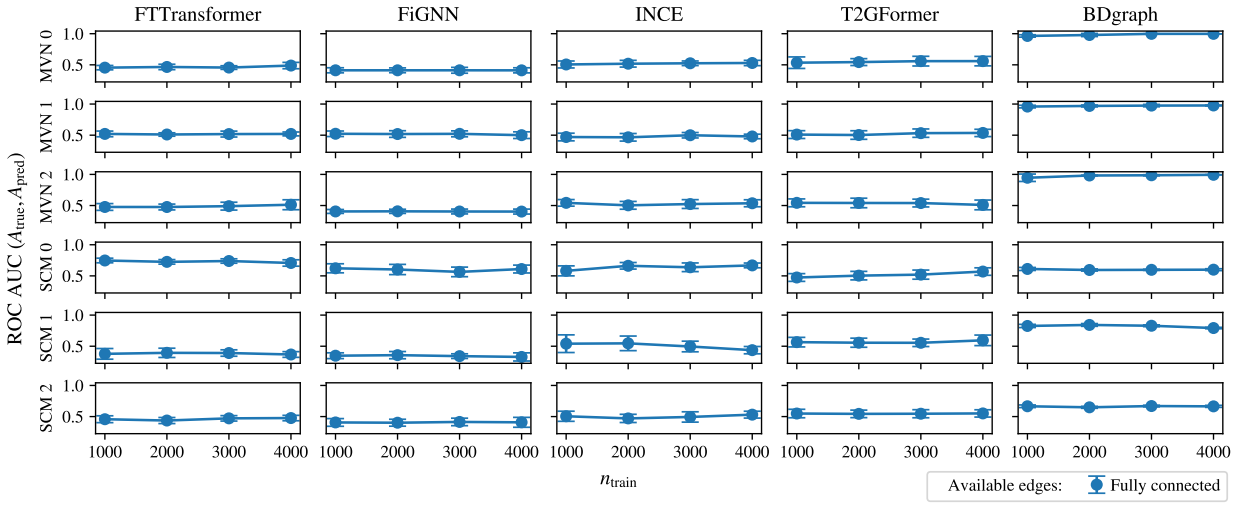

Figure 9: Graph quality in the form of the ROC AUC comparing the learned weighted adjacency matrix with the true binary one, for six different datasets. Most GTDL models have ROC AUC ≈ 0.5, which is random chance, indicating that they are not able to learn the feature interactions in any meaningful way. The PGM method BDgraph can learn the feature interactions. Error bars show standard deviation across seeds and cross validations. See Figure 4 for the results aggregated over the two dataset types.

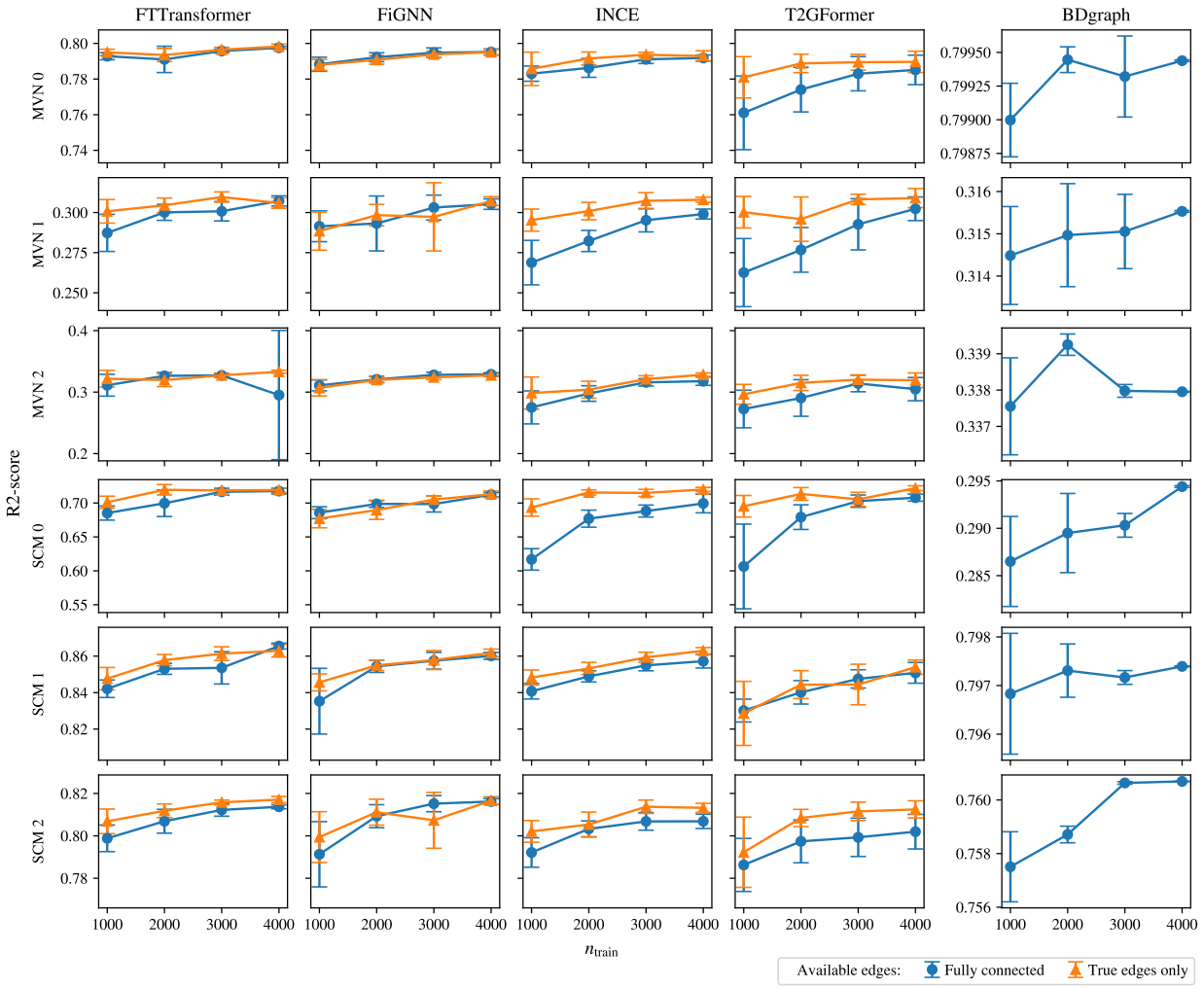

Figure 10: Predictive performance while varying the number of training samples n_{train}, for six different datasets. When the graph is pruned to its true edges, the predictive performance is, in most cases, better compared to the fully connected graph. This difference reduces as the number of training samples increases. Note the different scale for the y-axis. Error bars show standard deviation across seeds and cross validations. See Figure 5 for the results aggregated over the two dataset types.

D.2 Node-level versus graph-level models

In Figure 5, not all models benefit from pruning the graph. For FiGNN, the pruned and fully connected graphs have similar performance. This could be because FiGNN treats the task of predicting the target feature on a graph-level task, while the other models have a target token and treat it as a node-level task. As an example, we adapt the architecture from T2G-Former, which is by default a node-level model, to a graph-level model. Results on the predictive performance are shown in Figure 11. The graph-level model benefits less from pruning the graph than the node-level model. This indicates that the node-level models are more sensitive to the graph structure than the graph-level models.

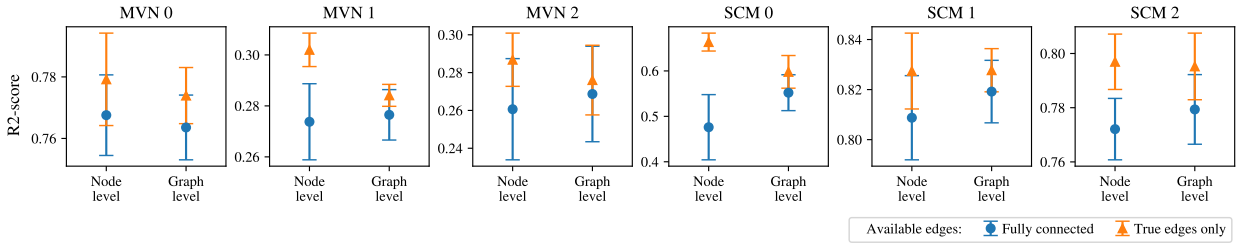

Figure 11: Node-level (default) versus graph-level adaptions of T2G-Former for multiple MVN and SCM datasets. The node-level adaptation benefits more from pruning the graph to the true edges. Error bars show standard deviation across seeds and cross validations.