Graph-based Tabular Deep Learning Should Learn Feature Interactions, Not Just Make Predictions

Elias Dubbeldam* **Reza Mohammadi** **Marit Schoonhoven** **Ilker Birbil**
Amsterdam Business School, University of Amsterdam

Abstract

Tabular data is characterized by complex, dataset-specific feature interactions. Graph-based tabular deep learning (GTDL) methods aim to address this by representing features and their interactions as a graph. However, existing methods predominantly optimize predictive accuracy, neglecting accurate modeling of the graph structure. In this work, we argue that GTDL should move beyond prediction-centric objectives and prioritize the explicit learning and evaluation of feature interactions. Using synthetic datasets with known ground-truth graph structures, we show that existing GTDL methods fail to recover meaningful feature interactions. Moreover, enforcing the true interaction structure improves predictive performance. This highlights the need for GTDL methods to prioritize quantitative evaluation and accurate structural learning.

1 Introduction

Deep learning has achieved remarkable success in domains such as natural language processing and computer vision, yet on tabular data it still struggles to compete with traditional, tree-based methods [9, 17]. Tabular data is characterized by heterogeneous features whose semantics differ and whose relationships (feature interactions) can be complex, indirect, and dataset-specific. By modeling these interactions, one incorporates an *inductive bias* (i.e., domain-specific principles encoded in the model's architecture [8, 3, 21]) that features interact differently. A natural way to encode this bias is with a graph, where nodes denote features and edges represent their interactions. Graph-based tabular deep learning (GTDL) methods seek to combine deep learning's expressive power with graph-structured feature representations. In particular, *feature* graph neural networks (GNNs), reviewed by [15], treat features as nodes and feature interactions as edges,[2] but how these interactions are modeled has not been extensively studied or evaluated, and current methods rarely assess whether the learned interactions correspond to meaningful relationships in the data.

This state of affairs raises important questions: Are GTDL models actually learning meaningful feature interactions, or are they merely optimizing predictive performance at the expense of interpretability? Without mechanisms to validate learned feature relationships, can we trust these models in practical, high-stakes use cases? Most critically, focusing exclusively on predictive accuracy of a GNN, risks encoding spurious interactions rather than capturing actual feature dependencies, undermining robustness, generalization, and explainability. We argue that GTDL methods must prioritize the learning, validation, and use of feature interactions as explicit modeling objectives. Current methods are predominantly prediction-centric, with little accountability for whether their learned structures reflect meaningful dependencies.

To support our argument, 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

*Correspondence to: e.f.dubbeldam@uva.nl

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

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: AI for Tabular Data.

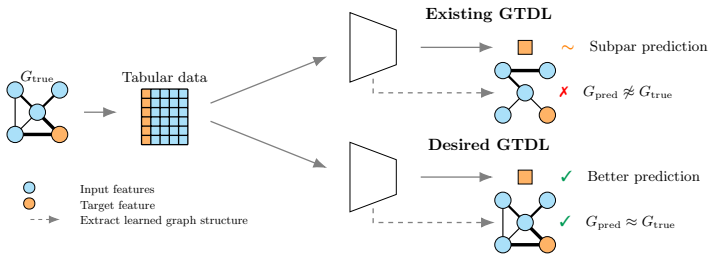


Figure 1: The true underlying graph structure generates tabular data. After training existing GTDL methods to predict the target feature, the extracted learned graph structure is not similar to the true graph structure. The predictive performance of GTDL methods improves when the extracted graph structure is accurate.

interactions. In Section 3, we argue that synthetic datasets and quantitative metrics are necessary to evaluate the graph structure. In Section 4, we show and discuss the results of controlled experiments to demonstrate our claims. Specifically, we highlight key reasons why explicitly modeling and evaluating feature graphs not only enhances predictive performance but also makes models more interpretable and reliable. In Section 5, we give a concluding discussion and propose future research directions.

2 Existing GTDL methods

In this section, we briefly review two approaches: attention-based methods and GNNs. We argue that both can be interpreted as GTDL methods, as they learn feature interactions on a graph structure.

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.[3] From trained GTDL methods, we can extract a weighted adjacency matrix $A_{\text{pred}} \in \mathbb{R}^{p \times p}$, where $0 \leq A_{ij} \leq 1$ indicates the strength of the interaction between features i and j, with $i, j \in \{1, \ldots, p\}$.

Attention-based methods Most recent tabular deep learning methods are attention-based [2, 11, 23, 13, 7, 10], from which FT-Transformer [7] has been established as a popular baseline. These methods are based on multi-head self-attention [25]. In most works, the attention map a is of size $p \times p$, where each cell a_{ij} corresponds to a feature interaction. Therefore, it can be used for interpretation and explaining the feature interactions. We note that when the attention map is averaged over the samples, heads, and layers, it can be interpreted as a weighted adjacency matrix, see Appendix A for more discussion on this. Therefore, we refer to such attention-based methods as *implicit* GTDL methods. Due to the nature of the attention map, they do model the feature interactions implicitly.

Tabular foundation models, TabPFN [10], TabICL [20] and LimiX [29], have feature-wise attention layers that could have been interpreted as implicit GTDL methods. However, TabPFN and LimiX encode group of features collectively rather than individually. TabICL incorporates rotary positional embedding [24] independent of the feature permutation, which alters the attention map. Therefore, we do not consider tabular foundation models in this work.

[3] For predictive models, the graph of interest is the CI graph. This is different from causal graphs, where the absence of an edge indicates no direct causal effect between two nodes.

Graph neural networks GNNs architectures operate directly on graph-structured data by propagating information between connected nodes [31]. Feature GNNs (such as FiGNN [16], T2G-Former [27], DRSA-Net [30], INCE [26], MPCFIN [28]) apply this paradigm to tabular data, modeling each feature as a node and explicitly learning feature interactions through message passing [15]. 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.

3 Evaluating feature interaction learning in GTDL

The development of GTDL is hindered by the fact that the learned graph structure is only evaluated heuristically. To solve this, we advocate in the next sections for the use of synthetic datasets and quantitative metrics to evaluate the learned graph structure.

Synthetic data to evaluate the 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, the feature interactions should be evaluated with *synthetic* datasets, which are close to real-world tabular data. Using synthetic data enables GTDL methods to compare the learned graph structure with the ground-truth underlying graph structure.

We adapt two existing data generation methods from the literature: (i) Multivariate normals (MVNs) are typically studied by probabilistic graphical models. We follow the default procedure of generating conditional multivariate data (as described in [18], for instance). (ii) Structural causal model (SCM) [19] are used to generate tabular data in tabular foundation models [10, 20, 29]. Both methods follow a three-step process to generate synthetic tabular data, more details are in Appendix B.

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 advocate that the learned feature interactions should be evaluated with the receiver operating characteristic area under curve (ROC AUC) [5] to quantify the learned graph structure. This could be done by comparing edge-wise (ignoring the diagonal) the true binary adjacency matrix A_{true} with the learned weighted adjacency matrix A_{pred}.

Pruning the feature interactions To highlight the importance of learning the feature interactions, we model the GTDL methods in two different settings. 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 learned graph structure within the network architecture.

4 Evidence for structure-aware learning in GTDL

To demonstrate the effect of using synthetic datasets, quantify the accuracy of the learned graph structure, and how the models perform when the graph is pruned to its true edges, we evaluate existing GTDL methods with a standard deep learning experiment. See Appendix B for details on the datasets and Appendix C for details on the experimental setup and . We compare all explicit GTDL methods that have publicly published code. That is, we compare FiGNN [16], T2G-Former [27] and INCE [26]. The remainder of the explicit GTDL methods, DRSA-Net [30], MPCFIN [28] and Table2Graph [32], have not published code. For implicit, attention-based methods, we take FT-Transformer [7] as an exemplary example.

Feature interactions The ROC AUC of the feature interactions is shown in Figure 2. 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 the adjacency matrix where there is, and where there is no true

edge. 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.

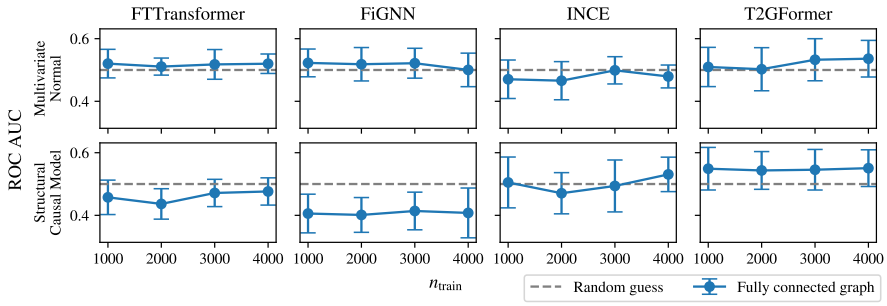


Figure 2: Graph quality in the form of the ROC AUC comparing the learned weighted adjacency matrix with the true binary one, for two different dataset types.

Predictive performance The R2 score of the prediction of the target feature is shown in Figure 3. The key takeaway is that, for most methods, **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. 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.

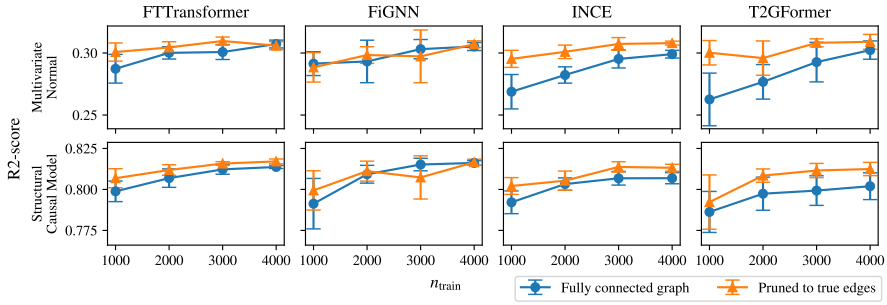


Figure 3: Predictive performance comparing a fully connected graph (default for GTDL methods) to the graph pruned to its true edges.

5 Conclusion

In this work, we argue that graph-based tabular deep learning (GTDL) must shift from a purely prediction-centric paradigm toward structure-aware modeling. Current GTDL approaches often produce graph structures used for interpretation, yet our analysis shows that these structures frequently fail to reflect the true interactions among features. This disconnect undermines interpretability, limits generalization, and erodes trust in model explanations. We advocate for the use of synthetic tabular datasets with known ground-truth graph structures, enabling the GTDL community to quantitatively assess whether models accurately capture the intended graph structure. 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 call for a new generation of GTDL models that incorporate structure-aware inductive biases, and support principled graph validation.

References

[1] 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*. KDD '19: The 25th ACM SIGKDD Conference on Knowledge Discovery and Data Mining. Anchorage AK USA: ACM, July 25, 2019, pp. 2623–2631. ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330701.

[2] Sercan O. Arik and Tomas Pfister. *TabNet: Attentive Interpretable Tabular Learning*. Dec. 9, 2020. DOI: 10.48550/arXiv.1908.07442. arXiv: 1908.07442. URL: http://arxiv.org/abs/1908.07442. Pre-published.

[3] 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*. Oct. 17, 2018. DOI: 10.48550/arXiv.1806.01261. arXiv: 1806.01261 [cs]. URL: http://arxiv.org/abs/1806.01261. Pre-published.

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. "Algorithms for Hyper-Parameter Optimization". In: *Advances in Neural Information Processing Systems*. Vol. 24. Curran Associates, Inc., 2011.

[5] Andrew P. Bradley. "The Use of the Area under the ROC Curve in the Evaluation of Machine Learning Algorithms". In: *Pattern Recognition* 30.7 (July 1997), pp. 1145–1159. ISSN: 00313203. DOI: 10.1016/S0031-3203(96)00142-2.

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

[7] Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. "Revisiting Deep Learning Models for Tabular Data". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 18932–18943.

[8] Anirudh Goyal and Yoshua Bengio. *Inductive Biases for Deep Learning of Higher-Level Cognition*. Aug. 1, 2022. DOI: 10.48550/arXiv.2011.15091. arXiv: 2011.15091 [cs]. URL: http://arxiv.org/abs/2011.15091. Pre-published.

[9] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. *Why Do Tree-Based Models Still Outperform Deep Learning on Tabular Data?* July 18, 2022. DOI: 10.48550/arXiv.2207.08815. arXiv: 2207.08815 [cs]. URL: http://arxiv.org/abs/2207.08815. Pre-published.

[10] 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". In: *Nature* 637.8045 (Jan. 9, 2025), pp. 319–326. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-024-08328-6.

[11] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. *TabTransformer: Tabular Data Modeling Using Contextual Embeddings*. Dec. 11, 2020. DOI: 10.48550/arXiv.2012.06678. arXiv: 2012.06678 [cs]. URL: http://arxiv.org/abs/2012.06678. Pre-published.

[12] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. Jan. 29, 2017. arXiv: 1412.6980 [cs]. URL: http://arxiv.org/abs/1412.6980. Pre-published.

[13] 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*. Vol. 34. Curran Associates, Inc., 2021, pp. 28742–28756.

[14] Gérard Letac and Hélène Massam. "Wishart Distributions for Decomposable Graphs". In: *The Annals of Statistics* 35.3 (July 2007), pp. 1278–1323. ISSN: 0090-5364, 2168-8966. DOI: 10.1214/009053606000001235.

[15] 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*. Jan. 4, 2024. arXiv: 2401.02143. URL: http://arxiv.org/abs/2401.02143. Pre-published.

[16] 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. Beijing, China and New York, NY, USA: Association for Computing Machinery, 2019, pp. 539–548. ISBN: 978-1-4503-6976-3. DOI: 10.1145/3357384.3357951.

[17] 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 15, 2024. DOI: 10.48550/arXiv.2305.02997. arXiv: 2305.02997 [cs]. URL: http://arxiv.org/abs/2305.02997. Pre-published.

[18] A. Mohammadi and E. C. Wit. "Bayesian Structure Learning in Sparse Gaussian Graphical Models". In: *Bayesian Analysis* 10.1 (Mar. 1, 2015). ISSN: 1936-0975. DOI: 10.1214/14-BA889. arXiv: 1210.5371 [stat].

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

[20] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. *TabICL: A Tabular Foundation Model for In-Context Learning on Large Data*. May 24, 2025. DOI: 10.48550/arXiv.2502.05564. arXiv: 2502.05564 [cs]. URL: http://arxiv.org/abs/2502.05564. Pre-published.

[21] 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, Sept. 10, 2024. DOI: 10.5463/thesis.738.

[22] Alberto Roverato. "Hyper Inverse Wishart Distribution for Non-decomposable Graphs and Its Application to Bayesian Inference for Gaussian Graphical Models". In: *Scandinavian Journal of Statistics* 29.3 (2002), pp. 391–411. ISSN: 1467-9469. DOI: 10.1111/1467-9469.00297.

[23] 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 2, 2021. DOI: 10.48550/arXiv.2106.01342. arXiv: 2106.01342 [cs]. URL: http://arxiv.org/abs/2106.01342. Pre-published.

[24] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. "RoFormer: Enhanced Transformer with Rotary Position Embedding". In: *Neurocomputing* 568 (Feb. 1, 2024), p. 127063. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2023.127063.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. Dec. 5, 2017. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762 [cs]. URL: http://arxiv.org/abs/1706.03762. Pre-published.

[26] Mario Villaizán-Vallelado, Matteo Salvatori, Belén Carro, and Antonio Javier Sanchez-Esguevillas. "Graph Neural Network Contextual Embedding for Deep Learning on Tabular Data". In: *Neural Networks* 173 (May 1, 2024), p. 106180. ISSN: 0893-6080. DOI: 10.1016/j.neunet.2024.106180.

[27] 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*. Vol. 37. AAAI'23/IAAI'23/EAAI'23. AAAI Press, Feb. 7, 2023, pp. 10720–10728. ISBN: 978-1-57735-880-0. DOI: 10.1609/aaai.v37i9.26272.

[28] Mang Ye, Yi Yu, Ziqin Shen, Wei Yu, and Qingyan Zeng. "Cross-Feature Interactive Tabular Data Modeling With Multiplex Graph Neural Networks". In: *IEEE Transactions on Knowledge and Data Engineering* (2024), pp. 1–15. ISSN: 1558-2191. DOI: 10.1109/TKDE.2024.3440654.

[29] 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*. Sept. 3, 2025. DOI: 10.48550/arXiv.2509.03505. arXiv: 2509.03505 [cs]. URL: http://arxiv.org/abs/2509.03505. Pre-published.

[30] 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". In: *IEEE Transactions on Knowledge and Data Engineering* 35.9 (Sept. 2023), pp. 9715–9727. ISSN: 1558-2191. DOI: 10.1109/TKDE.2023.3249186.

[31] 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*. Oct. 6, 2021. DOI: 10.48550/arXiv.1812.08434. arXiv: 1812.08434 [cs, stat]. URL: http://arxiv.org/abs/1812.08434. Pre-published.

[32] 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*. Thirty-First International Joint Conference on Artificial Intelligence {IJCAI-22}. Vienna, Austria: International Joint Conferences on Artificial Intelligence Organization, July 2022, pp. 2420–2426. ISBN: 978-1-956792-00-3. DOI: 10.24963/ijcai.2022/336.

A Attention map as adjacency matrix

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 define the set $\mathbb{R}_{[0,1]}$ as:

$$\mathbb{R}_{[0,1]} \coloneqq \{x \in \mathbb{R} \mid 0 \leq x \leq 1\} \subset \mathbb{R}. \tag{1}$$

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:

$$a_{ilhjk} = 0 \quad \forall \quad j = k. \tag{2}$$

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

$$A_{ilhjk} = a_{ilhjk} / \max_k(a_{ilhjk}). \tag{3}$$

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.

B Data generation

In this section, we describe the graph and data generation process of the two synthetic dataset approaches introduced in Section 3 and used in Section 4.

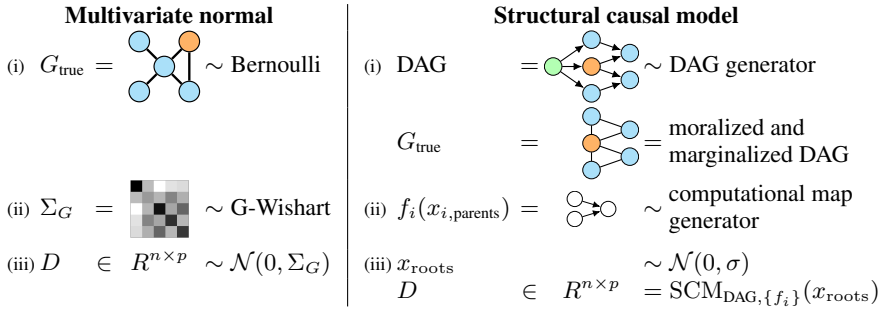


Figure 4: Two synthetic data generation pipelines. Both pipelines can roughly be divided into three steps. (i) Sample a graph structure; (ii) Sample feature interactions; (iii) Sample data given the graph and feature interactions. Nodes are colored with as • cyan input features x, • orange target feature y, and • green root nodes x_{roots}.

Multivariate normals. We follow the default procedure of generating conditional multivariate data, [18]:

(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 [22, 14], 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 $\mathcal{N}(0, \Sigma_G)$.

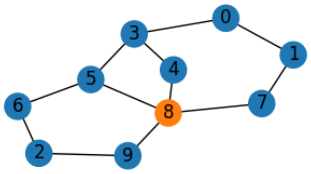
In our experiments, we have $p = 10$ nodes, and an edge inclusion probability of $P_{\text{edge}} = 0.267$. This results in the graph structures as depicted in Figure 5a.

Structural causal models. We follow a similar setup as [10] 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.

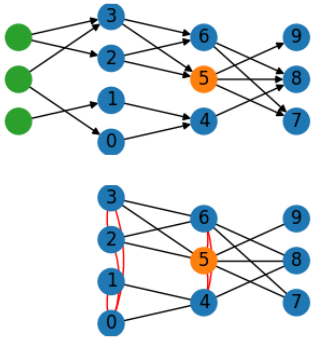
(i) Randomly sample a DAG, with n_{root} root nodes and p child nodes with a probability of P_{edge} of an incoming edge. The undirected graph structure G_{true} is obtained by moralizing and marginalizing the DAG [6]. Moralizing makes the graph undirected by dropping the direction of the existing edges 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 5b 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 1. A computational map defines how a child node i is computed from

[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}.



(a) MVN used in experiments.



(b) Top: directed acyclic graph (DAG) used in experiment. Bottom: Corresponding undirected graph structure G_{true} after moralization and marginalization. Moralized edges are depicted in red.

Figure 5: Graph structures used in experiments.

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

$$x_i = \text{clip}(\text{normalize}(f_i(x_{i,\text{parents}})) + \mathcal{N}(0, 0.5), -3, 3). \tag{4}$$

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 DAG that is used in Section 4 is shown in Figure 5b.

Table 1: Computational maps used in the SCM data generation process.

# 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	$(x_1 x_2 + x_3^2)/3$
	$-x_1^2 + x_2 x_3 + x_3$
	$(x_1 + x_2 + x_3) + x_1 x_3$

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 $x \in \mathbb{R}^{n \times (p-1)}$.

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 [9], 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. 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.

Training and evaluation. We minimize the mean squared error (MSE) loss function and optimize using Adam [12] 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 not 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) [4], a Bayesian optimization technique, within the Optuna library [1]. 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.

For all models, the search space and default values are taken from the original implementations. 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.