Cross-Feature Interactive Tabular Data Modeling With Multiplex Graph Neural Networks

Mang Ye, *Senior Member, IEEE*, Yi Yu, Ziqin Shen, Wei Yu, and Qingyan Zeng

***Abstract*—The rising popularity of tabular data in data science applications has led to a surge of interest in utilizing deep neural networks (DNNs) to address tabular problems. Existing deep neural network methods are not effective in handling two fundamental challenges that are inherent in tabular data: permutation invariance (where the labels remain unchanged regardless of element order) and local dependency (where predictive labels are solely determined by local features). Furthermore, given the inherent heterogeneity among elements in tabular data, effectively capturing heterogeneous feature interactions remains unresolved. In this paper, we propose a novel Multiplex Cross-Feature Interaction Network (MPCFIN) by explicitly and systematically modeling feature relations with interactive graph neural networks. Specifically, MPCFIN first learns the most relevant features associated with individual features, and merges them to form cross-feature embedding. Subsequently, we design a multiplex graph neural network to learn enhanced representation for each sample. Comprehensive experiments on seven datasets demonstrate that MPCFIN exhibits superior performance over deep neural network methods in modeling the tabular data, showcasing consistent interpretability in its cross-feature embedding module for medical diagnosis applications.**

***Index Terms*—Feature interactions, graph structure learning, tabular data.**

I. Introduction

In recent years, deep neural networks have excelled in various fields including computer vision and natural language processing. For example, models such as CNNs [1], RNNs [2], and Transformer [3] have demonstrated remarkable efficacy in handling diverse types of data like videos, images, text, and speeches. It is evident to observers that the success of these deep neural network architectures stems from their ability to effectively model the corresponding type of data, thus fully embodying their inherent characteristics. Empirical studies have shown that CNNs effectively model the structural relationships between image pixels by relying on efficient local inductive biases [4], while Transformers leverage the advantage of attention mechanisms to effectively model long-distance dependencies in sequential data [5].

In real-world applications, tabular data is one of the most commonly used forms of data, comprising independent samples (rows) with the same attributes (columns). It contains a wealth of crucial information for data-driven predictive analytics in various applications, such as recommendation systems [6], [7], [8], online advertising click-through rate prediction [9], medical treatment [10], and fraud detection [11]. While deep neural networks (DNNs) have achieved remarkable success, applying them to tabular data poses unique challenges. Unlike spatially or sequentially structured data like images or text, the distinctive characteristics of tabular data lie in the heterogeneity of its features. Two crucial characteristics of tabular data are the invariance of feature order permutations and the locality of feature dependencies [12], [13]. Unlike other data forms, such as text or images, where rearranging attribute positions or altering the sequence of sentences can result in semantic errors and recognition failures (Fig. 1(b)), the permutation of feature order in tabular data has no effect on the label column. This means that rearranging the attributes does not affect the overall outcome. Additionally, predictive labels may only depend on specific features rather than all. As illustrated in Fig. 1(c), we can see that the decisive features determining labels are highly correlated only with the highlighted red or green features. From the examples provided, it is evident that tabular data not only demonstrates local dependency but also exhibits absolute permutation invariance, distinguishing it from other data forms.

Existing deep methods for addressing tabular problems can be categorized into two types. The first type is hybrid models [14], [15], [16] that combine deep neural networks with classical machine learning methods, typically decision trees [17]. These models enable end-to-end deep learning training and inference, leveraging GPU or TPU acceleration. However, they often overlook the interaction among features [18], [19] and cannot handle high-dimensional complex data, making it difficult to mine the relationships between different elements. The second type is based on Transformer models [12], [17], [20], inspired by their success in text and visual data. These methods

Manuscript received 13 April 2024; revised 4 July 2024; accepted 1 August 2024. Date of publication 8 August 2024; date of current version 13 November 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62361166629, Grant 62176188, and Grant 62376200, in part by the Key Research and Development Program of Hubei Province under Grant 2022BAD175, in part by the Key Project of the Health Commission of Hubei Province under Grant WJ2023Z003, in part by the Research Fund of the Clinical Research Institute of Aier Eye Hospital Group under Grant AR2204D1, and in part by the Research Fund of Aier Eye Hospital Group under Grant AF2204D04, Grant AF2104D9. Recommended for acceptance by X. Zhu. *(Corresponding author: Wei Yu.)*

Mang Ye is with the National Engineering Research Center for Multimedia Software, School of Computer Science, Wuhan University, Wuhan, Hubei 430072, China, and also with the Taikang Center for Life and Medical Sciences, Wuhan University, Wuhan, Hubei 430072, China (e-mail: yemang@whu.edu.cn).

Yi Yu, Ziqin Shen, and Wei Yu are with the National Engineering Research Center for Multimedia Software, School of Computer Science, Wuhan University, Wuhan, Hubei 430072, China (e-mail: yuwei@whu.edu.cn).

Qingyan Zeng is with Aier Eye Hospital, Wuhan University, Wuhan, Hubei 430072, China.

Digital Object Identifier 10.1109/TKDE.2024.3440654

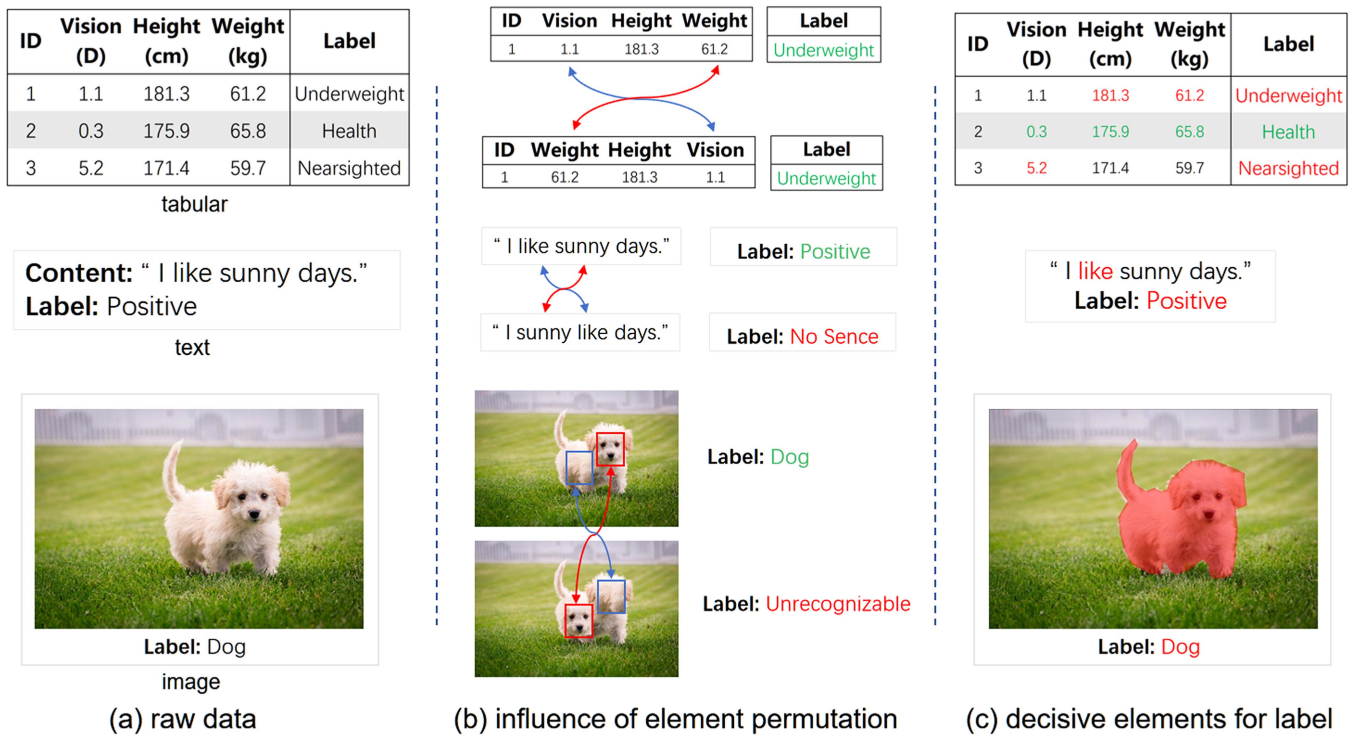


Fig. 1. Comparison of three examples for permutation invariance and feature local dependency. (a) Raw data for tabular data, text, and image. (b) Permutation Invariance: By exchanging the positions of elements in each data instance (such as reordering columns in tabular data, rearranging words in sentences, or adjusting the arrangement of pixels in an image), only tabular data demonstrates absolute permutation invariance. On the other hand, image data has the potential to achieve partial permutation invariance by exchanging non-decisive elements. This implies that even after permuting any element, the labels within tabular data remain unchanged. (c) Local Dependency: By presenting three examples that highlight the most influential features of labels in red or green, it is demonstrated that each type of data exhibits local dependency. This suggests that the labels are strongly correlated with only a subset of features.

employ attention mechanisms to handle heterogeneous tabular data, capturing local dependency and feature interactions [3]. However, Transformer-based models introduce redundant interactions and higher computational complexity, diminishing their performance [21]. Additionally, gradient boosting decision trees (GBDTs) [22] are recognized as state-of-the-art for tabular problems but only focus on single-sample modeling. In summary, existing methods for tabular data primarily exhibit two limitations: they either emphasize global feature interactions or individual sample modeling, disregarding the unique characteristics of tabular data, or they solely focus on local feature interactions, neglecting the important aspect of permutation invariance.

To improve the effectiveness of deep methods for solving tabular problems, it is crucial to find a suitable approach for modeling local feature dependencies and addressing the permutation invariance of feature orders. Interestingly, we observe that graph-structured data [23] exhibits properties similar to permutation invariance and local dependency found in tabular data. Graph nodes represent entities in the data and demonstrate permutation invariance, meaning that rearranging node order does not impact overall properties or relationships within the graph. Additionally, graph-structured data shows local dependency, with interactions and dependencies between nodes typically localized to specific subsets rather than involving the entire graph. These unique properties of *permutation invariance* and *local dependency* in graph-structured data provide valuable insights for utilizing Graph Neural Networks (GNNs) [23] to address the challenges in modeling tabular data. Therefore, we consider each column of tabular data as a node and connect the relationships between the columns with edges. This transformation converts each sample of the table into graph data. Subsequently, we incorporate with GNNs [23] to enable feature interaction through message passing between nodes, capturing the implicit relations among different elements.

However, leveraging GNNs directly to address tabular problems presents two significant challenges. First, since nodes are crucial elements in a graph and column attributes are the most important features in tabular data, the way we generate node embeddings using column attributes can significantly impact the efficiency and accuracy of graph message passing [24]. To enable effective feature interactions in graph neural networks (GNNs), existing methods often utilize a single column as the node embedding [13], [25], [26], [27], [28], [29]. In such cases, fully connected graph structures are typically employed to facilitate substantial interaction between columns. However, the use of a fully connected graph not only results in redundant information but also makes the model sensitive to noisy anomalies, which also indirectly proves that single column is not suitable for node embedding. As demonstrated in Fig. 1(c), determining a person's obesity level cannot solely rely on their height or weight alone. It is crucial to consider the appropriate ratio between height and weight. Thus, utilizing cross-features that encompass multiple columns for node embeddings aligns more closely with human intuition than relying solely on individual features. Additionally, cross-features inherently integrate multiple feature information, avoiding using a fully connected graph. Hence, it is more reasonable to consider cross-features as node embeddings. Worse still, as tabular data represents structured data with feature heterogeneity and the focus lies in the cross-feature information interaction between columns, applying traditional cross-feature methods like feature concatenation (sequentially concatenating the values of different feature columns into a longer feature vector) and direct feature interaction (capturing the correlations between different feature columns through interaction operations) [30], [31] for cross-feature interaction in tabular data inevitably leads to the curse of dimensionality and high computational complexity. Therefore, the conventional approaches for cross-feature interaction are not directly applicable to tabular data, highlighting the research gap in this aspect of cross-feature information interaction in tabular data. Second, the structure of the graph is another vital element in the analysis, as it effectively captures the interconnections between nodes. By representing the relationships between nodes, the graph structure plays a key role in illustrating the direction of message passing and identifying nodes with higher levels of interaction. The way we construct the graph structure has a significant impact on the performance of GNNs [32]. There are two existing ways for constructing graph structures: manually hand-crafted connectivity and automatically learning connectivity [33], [34]. The former strategy may result in redundant or missing information due to fixed connections, while the latter lacks human prior knowledge, potentially leading to slower learning efficiency or poor interpretability. Balancing these two approaches and achieving a graph structure that facilitates meaningful feature interaction while incorporating human prior knowledge remains an unresolved challenge.

In this paper, we propose a novel Multiplex Cross-Feature Interaction Network (MPCFIN) to model tabular data with GNNs, leveraging the advantages of hand-crafted and automatically learning connectivity. Specifically, MPCFIN consists of two main modules: graph construction and message passing. The graph construction module enables the end-to-end conversion of tabular data into graph-structured data, mining the underlying relations across multiple elements. Methodologically, this module automatically learns the most relevant features for each column with neural transform and concatenates them as cross-feature information. To enhance the representation, these concatenated features are then projected into a d-dimension space through linear layers to obtain node embeddings. Subsequently, a parallel multiplex graph structure is formed by combining hand-crafted connectivity with automatically learning connectivity. In the hand-crafted component, we employ the idea of minimally connected subgraphs to prevent redundancy and noise that can arise from a fully connected graph. This means we focus on identifying and including only the essential connections between features, reducing unnecessary complexity. On the other hand, the automatic learning component utilizes metric learning methods to formulate the graph structure, determining which features are more closely related or influential in the data context. This allows us to automatically uncover important connections and dependencies among the features without relying solely on manual feature engineering. By combining these two approaches, we can leverage the benefits of both, providing a more comprehensive and effective approach to feature engineering for tabular data.

In the message passing module, Interaction Network [13], [26] is employed for performing message passing in the aforementioned multiplex graph structure. By leveraging the Interaction Network, we can capture the dependencies and relationships between features, considering their interconnected nature within the multiplex graph structure. This facilitates the exchange of information and enables the model to learn complex patterns and interactions among features, enhancing the overall predictive capability. By extracting embedding representations of the CLS (classification) nodes from both graphs and concatenating them, our framework showcases exceptional performance, surpassing various deep neural network (DNN) methods. Compared to traditional DNN methods, the ability of our framework to leverage the concatenated embeddings provides a more nuanced and informative representation.

Overall, the main contributions of our work are as follows:

- We propose a Multiplex Cross-Feature Interaction Network (MPCFIN) to effectively address the permutation invariance and local dependency in tabular data. Through the use of multiplex graphs and a dedicated message passing module, MPCFIN effectively models the interactions and dependencies between features, resulting in improved predictive performance compared to existing approaches.
- We introduce a novel adaptive cross-feature interaction method, which automatically learns the most relevant features for each attribute in a tabular dataset. By leveraging cross-feature information, our method effectively models the interactions between features, demonstrating superior predictive performance with strong interpretability.
- We use a multiplex graph structure to effectively integrate hand-crafted connectivity and automatically learning connectivity. This offers a valuable approach to refer for addressing the graph connectivity issues. Experiments on seven datasets, including one clinical dataset, have demonstrated the superiority of our proposed solution.

II. Related Work

A. Tabular Modeling

For tabular problems, models based on gradient boosting decision trees (GBDTs) algorithms such as XGBoost [35], CatBoost [36], and LightGBM [37] have long been recognized as state-of-the-art. Regardless of the characteristics of the tabular data, including sample size, number of features, or amount of missing values, GBDTs-based models often demonstrate powerful performance compared to other models. Prior to the emergence of GBDTs, more common approaches generally included Support Vector Machines (SVM) and Multilayer Perceptrons (MLP). However, these classical methods model the interaction between features in a rather simplistic manner, and even SVM struggles with sparse data [38]. Nevertheless, the limitations of these non-neural network models based on GBDTs are also apparent. They lack the capability to handle the challenge of permutation invariance and only focus on single-sample modeling.

Compared to tree-based models, DNNs possess the strong ability to handle high-dimensional complex data and powerful automatic feature learning capabilities [39]. An increasing number of studies have begun to utilize deep learning methods to address tabular-related problems [17]. Next, we will categorize these deep learning methods into three classes and elaborate on each of them.

Hybrid Models: As mentioned earlier, hybrid models transform tabular data and combine deep neural networks with classical machine learning methods, typically decision trees. Since neural networks mostly utilize gradient descent for optimization, while decision trees and other machine learning methods are often based on probabilistic analysis, hybrid models can be further categorized into fully differentiable models and partially differentiable models [17]. Fully differentiable models achieve fully-differentiable end-to-end deep learning training and inference through gradient descent optimizers. An example of a fully differentiable model is the differentiable oblivious decision trees known as NODE [40], which draws inspiration from the successful CatBoost [36] framework. NODE employs the entmax transformation and soft splits to achieve a fully differentiable framework. In NODE, the concept of differentiable decision trees [41] is combined with multi-layer hierarchical representations, resulting in competitive performance as GBDTs approaches. Partially differentiable models combine non-differentiable methods with deep neural networks. An example is the TabNN [42] framework, which distills the knowledge from GBDTs to retrieve feature groups. These feature groups are then clustered and used to construct neural networks. However, in many data scenarios with outliers and missing values, hybrid models lack the ability to capture interaction between features.

Transformer-Based Models: With the sweeping success of the Transformer in various applications, it has inspired a multitude of recent research endeavors proposing diverse approaches to handling heterogeneous tabular data using attention mechanisms. TabNet [43], as one of the pioneering models to employ attention mechanisms for addressing tabular problems, combines self-attention and decision tree structures, thereby possessing the capability of feature selection and feature importance evaluation. To better represent categorical features within tables, in TabTransformer [44], the authors first utilize specialized feature encoders to encode each categorical feature, followed by employing multi-layer Transformer encoders for interaction of these categorical features. Finally, these features are concatenated with continuous features to serve as embedding feature vectors for the tabular data. However, TabTransformer solely focuses on the interaction among categorical features, neglecting the interaction between categorical and continuous features. SAINT [12], on the other hand, effectively addresses this issue. SAINT utilizes attention mechanisms to enable interactions between categorical and continuous features, as well as between columns and rows within the table. This approach not only facilitates a thorough interaction between categorical and continuous features but also allows missing rows in tables with many missing values to be completed by referencing other rows using attention mechanisms. However, Transformer-based models introduce extra noise and redundancy, and they face challenges in accurately modeling the permutation invariance of tabular data.

GNNs-Based Models: Using Graph Neural Networks (GNNs) to model tabular problems has recently gained popularity. The primary process can be roughly divided into three steps: first, modeling the tabular data as a graph structure; second, utilizing graph neural networks to propagate messages within the graph; and finally, extracting node embeddings for downstream tasks. Such models can be categorized into two types: graph modeling based on the structure between samples (rows) and graph modeling based on the structure between attributes (columns). Sample-based modeling refers to treating each row in the table as a node, with the attributes of each row serving as node features, and then employing GNNs to propagate messages between rows. An example of this is TabGNN [45]. TabGNN models the table as multiple graph structures, leveraging human prior knowledge and the commonalities among certain columns. Attention mechanisms are then used to interactively process the outputs of each graph and obtain the final representation. However, according to [12], [44], [46], the most challenging aspect of tabular data lies in resolving the relationships between heterogeneous features. The effectiveness of a model heavily relies on whether a good way to interact with features exists. Therefore, instead of focusing on sample-based modeling, we concentrate on graph modeling based on the structure between attributes (columns). In INCE [13], the authors model individual feature columns as nodes and introduce a CLS [47] node, effectively transforming the tabular data into a fully connected graph. In T2G-FORMER [28], the authors construct graph structures from two perspectives: soft selection and hard selection. These two perspectives are fused to obtain the relationship graph between features, and attention mechanisms are then applied to learn representations on this relationship graph. Similar to the T2G-FORMER approach, DRSA-Net [38] also constructs graph structures from two perspectives and fuses them to obtain the relationship graph. However, instead of using attention mechanisms to learn representations, DRSA-Net employs GNNs. Nevertheless, these methods do not address the cross-feature problem from a human reasoning perspective.

B. Graph Structure Construction

How to define a graph structure is crucial for feature interaction as it determines the effectiveness of the resulting embedding representation. The most common approach to constructing a graph structure is using hand-crafted connectivity, which means manually determining the structure of the graph and cannot be changed once determined. In TabGNN [45], the authors utilize human prior knowledge (such as the assumption that individuals with the same education level may have similar income) to pre-determine potential connections between samples and then construct edges between these samples. Fully connected graph structures are used in DeepFM [48] for multiplicative feature interaction. Similarly, DCN [49] and DCNv2 [50] employ fully connected graphs combined with deep neural networks to learn complex features with high-order interactions. Additionally, INCE [13] also uses a fully connected graph to represent the relationships between the CLS node and feature nodes. Apart from hand-crafted connectivity, a widely used approach for graph construction is automatically learning connectivity, also known as graph structure learning. Graph structure learning aims to train a graph structure learner to model the connectivity of edges and optimize the graph structure relationships between nodes. According to [51], existing graph structure learning methods can be roughly classified into three categories: (1) metric-based approaches, (2) neural approaches, and (3) direct approaches. The first method relies mainly on matrix operations to calculate the relevance between pairs of nodes based on their embeddings. Common matrix operations include inner product and cosine similarity. For example, in EGG-GAE [27], the euclidean distance between nodes is computed to determine the graph's connectivity. In IDGL [52], cosine similarity is used to calculate edge weights in the graph. Unlike metric-based approaches that directly compute the relevance between node pairs, neural approaches first learn node embeddings through neural networks and then perform operations with trainable matrices to obtain a new adjacency matrix. For example, in GLN [53], the neural network learns an embedding matrix $H_{local}^{(l)}$, which is then multiplied by trainable parameters α_l and $M^{(l)}$ to obtain the adjacency matrix $A^{(l+1)}$. In contrast to the previous two methods, direct approaches do not require any operations and directly introduce a trainable parameter θ to represent the predicted adjacency matrix. Some typical examples include GLNN [54], ProGNN [55], BGCN [56], and so on. However, hand-crafted connectivity can lead to redundant or missing information with fixed connections, while automatically learning connectivity lacks human prior knowledge, which may result in slower learning or poor interpretability. Therefore, we propose multiplex graph modeling to effectively address the limitations of these two methods.

III. Problem Formulation and Preliminaries

Tabular data refers to structured data with N rows and M columns, where each row represents a sample and each column represents a feature. Our work primarily focuses on two supervised prediction problems in tabular data: regression and classification. Mathematically, given a tabular dataset $\mathcal{D} = \{x_i^j, y_i\}_{i\in[1,N],j\in[1,M]}$, where x_i^j represents the j-th column feature of the i-th row sample, and y_i represents the label (continuous values for regression and discrete values for classification) of the i-th row sample. We denote the feature set of the tabular as $\mathcal{X} = \{x_i|i \in [1, N]\}$ and the label set as $\mathcal{Y} = \{y_i|i \in [1, N]\}$. Furthermore, to differentiate between numerical and categorical features, we use $\mathcal{S}_{num}$ to represent the index set of numerical features and $\mathcal{S}_{cat}$ to represent the index set of categorical features. The goal of tabular problems is to optimize a function $\hat{y_i} = f(x_i)$, minimizing the following equation:

$$\min \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i), \tag{1}$$

where $\mathcal{L}$ is the loss function. For regression problems, $\mathcal{L}$ can be a loss function such as mean squared error, while for classification problems, it can be cross-entropy loss function. Since we employ GNNs for tabular modeling, we shall subsequently provide a formal representation of the pertinent concepts associated with GNNs.

We propose Multiplex Cross-Feature Interaction Networks (MPCFIN) to model each row sample of the table, as illustrated in Fig. 2. For a sample (x_i, y_i), each column feature x_i^j is mapped to latent vector $h_i^j \in \mathbb{R}^d$ through the cross-feature embedding module. Subsequently, each latent vector is considered as a node, with a CLS node $h_i^0 \in \mathbb{R}^d$ being added to form a graph structure $\mathcal{G}_i = \{\mathcal{V}, \mathcal{A}\}$, where $\mathcal{V} = \{v_0, v_1, \ldots, v_M\} = \{h_i^0, h_i^1, \ldots, h_i^M\}$ represents the set of nodes. Here, v_0 denotes the CLS node, and h_i^0 is the embedding vector of the CLS node. The adjacency matrix $\mathcal{A} \in \{0, 1\}^{(M+1)\times(M+1)}$ is shared across all samples. Through the adjacency matrix $\mathcal{A}$, we can obtain the set of edges $\mathcal{E} = \{e(v_i, v_j)|v_i, v_j \in \mathcal{V} \wedge \mathcal{A}_{ij} = 1\}$ and the set of neighboring nodes $\mathcal{N}(i) = \{j|v_j \in \mathcal{V} \wedge \mathcal{A}_{ij} = 1\}$.

Given that we are employing GNNs to model tabular data, here we provide a concise overview of the message passing mechanism's underlying principles for enhanced comprehension and readability in subsequent sections.

Message Passing Mechanisms are a general framework for supervised or semi-supervised learning on graphs, which abstracts the commonalities between various GNNs. Specifically, an initial message function defines how messages are generated and updated for each node on the given graph $\mathcal{G}_i$. Subsequently, an aggregation function is designed to aggregate the messages received from source nodes onto the target nodes. Lastly, an update function is incorporated to learn the node representations for each node. We formalize this process as follows:

$$\begin{aligned} h_i^{agg} &= \bigoplus_{k\in\mathcal{N}(j)} \phi^{(t)}\left(h_i^{j(t-1)}, h_i^{k(t-1)}\right), \\ h_i^{j(t)} &= \gamma^{(t)}\left(h_i^{j(t-1)}, h_i^{agg}\right), \end{aligned} \tag{2}$$

where $h_i^{j(t)}$ represents the representation of node h_i^j at the t-th iteration and $h_i^{j(0)}$ denotes the raw features obtained through cross-feature embedding. ϕ, $\oplus$, and γ respectively denote the message function, aggregation function, and update function. The aggregation function is typically a choice between sum, mean, max, and attention functions, while the message function and update function are usually differentiable functions such as MLPs. After T iterations, the representation vector of the CLS node can be extracted for prediction tasks.

IV. Multiplex Cross-Feature Interaction Networks

As shown in Fig. 2, the proposed Multiplex Cross-Feature Interaction Networks (MPCFIN) consist of two parts: the Cross-Feature embedding module and the Multiplex GNN module. In the preceding sections, we have provided a concise overview of MPCFIN. This section presents a detailed description of each module and offers comprehensive elucidations on their specifics,

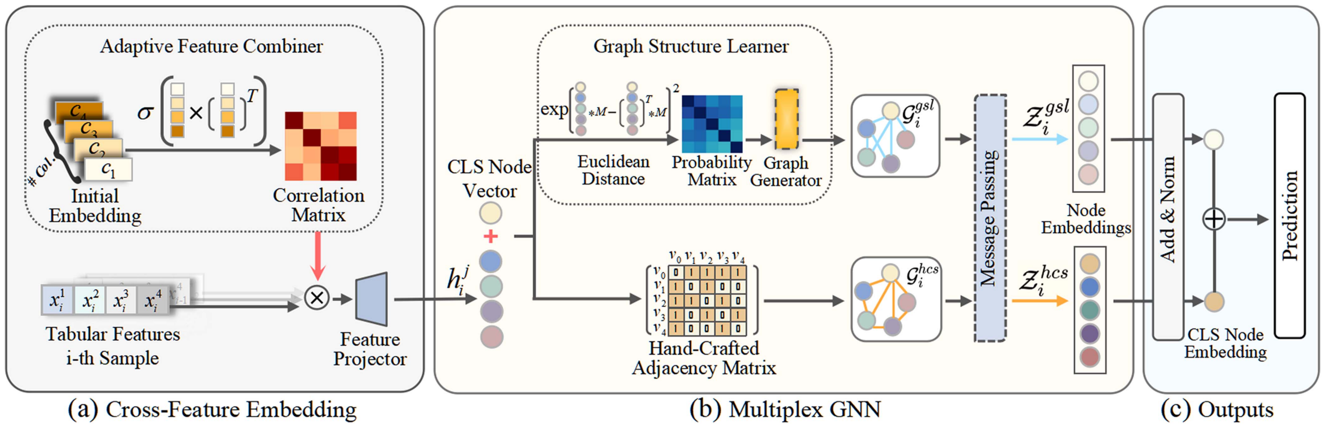


Fig. 2. The overall pipeline of MPCFIN. It consists of three main modules: Cross-Feature Embedding module, Multiplex GNN module, and Outputs module. (a) The detailed framework of the Cross-Feature Embedding Module. Its inputs are tabular features. (b) The comprehensive architecture of the Multiplex GNN module. It comprises two main blocks: an Auto-Learning block in the upper half and a Hand-Crafted block in the lower half. (c) The outputs module details. It extracts the embedding vectors of CLS node and then uses them for downstream prediction task.

consistently utilizing all mathematical definitions from the previous section. In addition, we will introduce a comprehensive complexity analysis to evaluate the computational efficiency of the MPCFIN model.

A. Cross-Feature Embedding Module

By employing the cross-feature embedding module illustrated in Fig. 2(a), all original heterogeneous features, denoted as x_i^j, are fused with cross-feature information and projected into a homogeneous and dense latent space $h_i^j \in \mathbb{R}^d$. To automatically learn the most relevant features for each feature and concatenate them to form cross-features, we propose an adaptive features combiner (AFC) to accomplish this task. In AFC, we first unify numerical and categorical features by employing the following transformation:

$$x_i^j = \begin{cases} f_n(\mathbf{x}_i^j), & \text{if } j \in \mathcal{S}_{num}, \\ f_d(\mathbf{x}_i^j), & \text{if } j \in \mathcal{S}_{cat}. \end{cases} \tag{3}$$

where $\mathbf{x}_i^j$ represents the raw features, f_n, f_d are normalized and discretized functions respectively, and we utilize the StandardScaler function as f_n in our experiments. Regarding the f_d function, various existing methods are dedicated to handling categorical features, such as [44]. Choosing an appropriate approach to handle categorical features can significantly impact the model's performance. Therefore, we explored three different methods: LabelEncoder, One-Hot encoding, and directly mapping categorical features to a d-dimensional space using an embedding layer. After comparing the results, we ultimately selected the LabelEncoder function as the final f_d. Next, to obtain a correlation adjacency matrix between columns, we employ learnable column embeddings $E_{col} = (c_1, c_2, \ldots, c_M) \in \mathbb{R}^{d\times M}$ to represent the embedding vectors of each column. We then calculate the possibility score matrix for potential correlations between columns using the following formula:

$$P[i,j] = p_r(c_i, c_j) = \frac{c_i^T c_j}{\|c_i\|_2 \|c_j\|_2}, \tag{4}$$

where $P \in \mathbb{R}^{M\times M}$ is the possibility score matrix, p_r is the possibility score calculation function, and to ensure smooth training parameters, we apply L_2 normalization. By using (4), we obtain the probability score matrix P. It is worth noting that here, our probability score matrix P is symmetric. However, it can also be a non-symmetric matrix by considering the relationships between columns as a directed graph and initializing two E_{col}, where one is treated as the source node embedding and the other as the target node embedding. Next, we calculate the correlation adjacency matrix $A_c \in \mathbb{R}^{M\times M}$ using the following formula:

$$A_c = [\sigma(P + b) > T], \tag{5}$$

where σ represents the element-wise activation function such as the sigmoid function, b is a learnable bias like PReLU [57], and T is a predefined constant threshold that controls the number of related features. In this way, the correlation between the i-th and j-th columns is stored in $A_c[i,j]$. Based on A_c and $\mathcal{X} = \{x_i | i \in [1,N]\}$, we can obtain the cross-feature representation $x_i'^j$ as follows:

$$x_i'^j = \text{Concat}\left(x_i^j, x_c^j\right), \tag{6}$$

$$x_c^j = f_c(\mathcal{X} \times A_c, j), \tag{7}$$

$$f_c(M, j) = M[:, j], \tag{8}$$

where $x_i'^j$ represents the cross-feature embedding of x_i^j after undergoing the AFC transformation, x_c^j is the fusion feature composed of the features that are most correlated with x_i^j, f_c is a simple selection function. It is important to note that the goal of AFC is to acquire a correlation matrix that represents intricate relationships among features, such as the underlying link between height and weight. By capturing correlations between columns, AFC ascertains which features are the most closely related and influential within the data context. This is essential for understanding feature interactions and guiding feature transformation processes within the model. Column embeddings are initialized randomly instead of using actual tabular data for three specific reasons: 1) The substantial volume of tabular data leads to unmanageable input dimensions for embeddings, complicating computations. 2) Similar to generative models such as GANs and VAEs, randomly initialized vectors help avoid overfitting and enhance robustness. 3) Hidden relationships between features are frequently not obvious in real data and can only be revealed through exploration strategies similar to reinforcement learning. In summary, opting for random initialization of column embeddings is advantageous for obtaining a more realistic correlation matrix and for exploring the hidden relationships between features.

Up to this point, we have obtained the cross-features $x_i^{'j}$ for each feature. Next, we further utilize a feature projector to map each cross-feature into a dense and homogeneous d-dimension latent space. The projector is an MLP structure represented by the following formula:

$$h_i^j = \sigma_1\left(b^j + x_i^{'j} \cdot W^j\right), \tag{9}$$

where σ_1 is the non-linear activation function such as ReLU, b^j is the learnable bias, $W^j \in \mathbb{R}^d$ is a learnable vector. The reason for adding superscript j to b and W is that each feature uses an unshared MLP. This allows for a more advantageous representation of each cross-feature. Through the above steps, all the original heterogeneous feature x_i^j is projected on the same dense and homogeneous d-dimension latent space, and h_i^j can be used as the node embedding in graphs.

B. *Multiplex Graph Neural Network*

In this part, we have devised a multiplex graph neural network for modeling the tabular into double graph structures. Subsequently, both graph structures pass through a graph neural network, more specifically, the Interaction Network [26]. Finally, the CLS node embeddings are extracted from both graphs and fused as input for the prediction task, as depicted in Fig. 2(b) and (c).

To go into the details of this module, drawing inspiration from [1] and [47], our objective is to acquire a global vector representation for each sample relative to the sample itself. To achieve this, we begin by incorporating a learnable embedding for each sample, namely the CLS node vector $h_i^0 \in \mathbb{R}^d$. This embedding is initialized using Xavier uniform initialization and merged with other features embedding, resulting in a set of nodes $\mathcal{V} = \{v_0, v_1, \ldots, v_M\} = \{h_i^0, h_i^1, \ldots, h_i^M\}$, where v_0 represents the CLS node. Moving forward, we employ two blocks to construct the multiplex graphs, namely auto-learning block and hand-crafted block, as shown in the upper half and the lower half of Fig. 2(b) respectively.

Hand-Crafted Block: This approach uses hand-crafted connectivity to construct a graph as illustrated in hand-crafted block shown in the lower half of Fig. 2(b). Since hand-crafted connectivity involves manually defining the graph structure, it is necessary to pre-construct the graph structure based on prior knowledge. Before we get into the details, to distinguish the CLS node from other normal nodes, we define the index set for the CLS node as $\mathcal{S}_{cls} = \{0\}$ and the index set for normal nodes as $\mathcal{S}_{nor} = \{1, 2, \ldots, M\}$. With the aforementioned definitions in place, we can now proceed to a formalized description of the details. First, considering that the CLS node serves as a global node while the other normal nodes represent local feature nodes, it is reasonable to establish bidirectional connections between the CLS node and each local feature node. By performing the following (10) operation, we can obtain an adjacency matrix $\mathcal{A}^{cls} \in \{0,1\}^{(M+1)\times(M+1)}$ that fulfills the aforementioned requirement:

$$\mathcal{A}^{cls}[i,j] = \begin{cases} 1, & \text{if } i \in \mathcal{S}_{cls} \vee j \in \mathcal{S}_{cls}, \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

In addition to the connections between the CLS node and the normal nodes, we also need to consider the connections between the normal nodes themselves. Since we have already taken into account the cross-feature information, we can construct the connections between normal nodes using a minimal connected subgraph approach, in other words, sequentially connecting the normal nodes. Based on the minimal connected subgraph and $\mathcal{A}^{cls}$, we can construct the hand-crafted adjacency matrix $\mathcal{A}^{hcs} \in \{0,1\}^{(M+1)\times(M+1)}$ for the hand-crafted structure according to the following formula:

$$\mathcal{A}^{hcs} = f_{\text{mcsg}}\left(\{v_i | i \in \mathcal{S}_{nor}\}\right) \cup A^{cls}, \tag{11}$$

where f_{mcsg} is a function that obtains the adjacency matrix of minimal connected subgraph. Based on $\mathcal{V}$ and $\mathcal{A}^{hcs}$, we can derive the ultimate hand-crafted graph $\mathcal{G}_i^{hcs} = \{\mathcal{V}, \mathcal{A}^{hcs}\}$.

Auto-Learning Block: To automatically construct the graph structure between nodes, we employed a graph structure learner (GSL) to accomplish this task, as seen in the upper half of Fig. 2(b). GSL utilizes a metric-based approach to learn the graph structure, leveraging the embedding vectors h_i^j of node pairs to compute the weights of edges between them. Specifically, we initially compute the probability matrix P^{gsl} that represents the relationship between pairs of nodes, where the j-th,k-th element is computed as:

$$P^{gsl}[j,k] = \exp^{\left\|h_i^j - h_i^k\right\|^2}, \tag{12}$$

where the j-th,k-th element represents the probability of sampling edge $e(v_j, v_k)$ in the output graph. Different with correlation matrix, the probability matrix defines the connectivity and relationships between nodes in a graph structure while the correlation matrix represents implicit feature-level relationships. This matrix guides the construction of graph representations, enabling the model to effectively capture node interactions and dependencies in a graphical format. The next step involves obtaining the adjacency matrix of an undirected graph using the probability matrix P^{gsl}. To achieve this, we employ a graph generator to effectively generate adjacency matrix from probability matrix as shown in Fig. 3. In graph generator, to optimize computational complexity, we first derive a strictly soft upper triangular adjacency matrix $\hat{A}^{gsl}$. In this process, we utilize a triangular mask matrix to sample P^{gsl} and then apply the Gumbel-Softmax trick [58] to obtain $\hat{A}^{gsl}$. The formula is as follows:

$$\hat{A}_{ij}^{gsl} = \frac{1}{1 + \exp\left(^{\left(\log\left(P_{ij}^{gsl} \circ \mathcal{M}_{ij}\right) + G_{ij} \circ \mathcal{M}_{ij}\right)/\tau}\right)}, \tag{13}$$

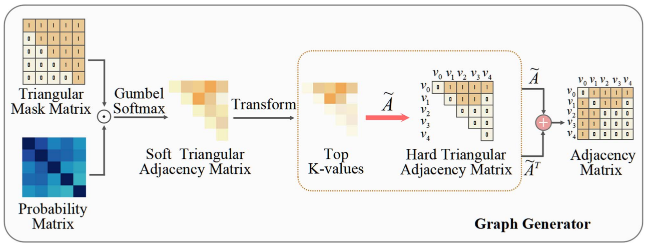


Fig. 3. The specific operations in graph generator of Fig. 2(b).

where $\mathcal{M}$ represents the upper triangular masking matrix, $\circ$ denotes the Hadamard product, G follows the Gumbel(0,1) distribution, and τ represents a temperature hyperparameter. However, the current matrix $\hat{A}^{gsl}$ is a soft adjacency matrix, where each element represents a specific value indicating the weight of the edge. The graph structure remains fully connected at this stage and does not provide sufficient specificity. Therefore, the subsequent step is to transform the soft adjacency matrix into a hard adjacency matrix, where each element strictly adheres to the binary values of 0 and 1. We proceed by selecting the top k values from each row of $\hat{A}^{gsl}$ where k is a hyperparameter. Subsequently, we set the positions corresponding to these k values as 1, while the remaining positions are set as 0. Finally, we obtain the final adjacency matrix using the following method:

$$\mathcal{A}^{gsl} = \tilde{A}^{gsl} + \left(\tilde{A}^{gsl}\right)^T,$$
$$\tilde{A}^{gsl} = f_{\text{top\_k}}\left(\hat{A}^{gsl}\right), \tag{14}$$

where the $f_{\text{top\_k}}$ function is employed to accomplish the aforementioned functionality of transforming soft adjacency matrix. Based on $\mathcal{V}$ and $\mathcal{A}^{gsl}$, we can derive the ultimate learned graph $\mathcal{G}_i^{gsl} = \{\mathcal{V}, \mathcal{A}^{gsl}\}$. In the aforementioned graph generator process, the probability matrix, soft matrix, and hard matrix are noteworthy. The probability matrix guides the construction of the graph by defining the initial probabilities of edge connections between nodes. It serves as the foundational element for determining relationships and connectivity within the graph structure. The soft matrix refines the initial probabilities from the probability matrix into weighted connections between nodes. A triangular mask matrix is used to generate the soft matrix. This mask samples probabilities from the probability matrix before applying the Gumbel-Softmax trick. The hard matrix converts the weighted connections of the soft matrix into binary connections, indicating whether an edge exists between nodes. This simplifies the graph structure, making it easier for further processing and analysis.

After constructing the graph structure, $\mathcal{G}_i^{hcs}$ and $\mathcal{G}_i^{gsl}$ are individually inputted into a graph neural network to update and propagate node representations. To be more precise, the Interaction Network is employed. The Interaction Network initializes the mean embedding of each pair of nodes connected by an edge as the embedding of that particular edge. Subsequently, it concatenates the embedding of each edge with the embeddings of its connecting nodes, as well as concatenates the embedding of each node with the embeddings of its connected edges. Following this step, MLP is utilized by the network to perform message passing and aggregation operations described in (2). Finally, updated embeddings for both edges and nodes are obtained by adding hidden vectors resulting from MLP to their original vectors. The algorithmic process of the whole Interaction Network is illustrated in Algorithm 1.

Algorithm 1: Interaction Network.

```
Require:
    The graph $\mathcal{G}_i = \{\mathcal{V}, \mathcal{A}\}$;
    The edges $\mathcal{E} = \{e(v_i, v_j) | v_i, v_j \in \mathcal{V} \wedge \mathcal{A}_{ij} = 1\}$;
    The neighbors of nodes
    $\mathcal{N}(i) = \{j | v_j \in \mathcal{V} \wedge \mathcal{A}_{ij} = 1\}$;
Ensure:
    Embedding of $e(v_m, v_n)$ is initialized to
      $\text{mean}(h_i^m, h_i^n)$;
1:  Set the number of iterations to $T$;
2:  for $t = 1, 2, \ldots, T$ do
3:    Set temporary tuples $e'$and $h_i'$;
4:    for $e(v_m, v_n) \in \mathcal{E}$ do
5:      $e'(v_m, v_n) =$
        $\text{MLP}_1(\text{Concat}(h_i^m, h_i^n, e(v_m, v_n)))$;
6:    end for
7:    for $h_i^j \in \mathcal{V}$ do
8:      $h_i^{'j} = \text{MLP}_2(\text{Concat}(h_i^j, \sum_{k \in \mathcal{N}(j)} e(v_k, v_j))$;
9:    end for
10:   for $e(v_m, v_n) \in \mathcal{E}$ do
11:     $e(v_m, v_n) = e(v_m, v_n) + e'(v_m, v_n)$;
12:   end for
13:   for $h_i^j \in \mathcal{V}$ do
14:     $h_i^j = h_i^j + h_i^{'j}$;
15:   end for
16:   Free temporary tuples $e'$and $h_i'$;
17: end for
18: Output: Nodes embedding $\mathcal{Z}_i = \{z_i^j | z_i^j = h_i^j\}$.
```

After passing through the Interaction Network, the final node embeddings $\mathcal{Z}_i^{gsl}$ and $\mathcal{Z}_i^{hcs}$ of the two graphs are read out. Next, the global representations $z_i^{gsl} = \mathcal{Z}_i^{gsl}[0]$ and $z_i^{hcs} = \mathcal{Z}_i^{hcs}[0]$ of the CLS nodes in the two graphs are extracted separately. To prevent gradient vanishing or exploding issues that may hinder model convergence during the learning process, we address z_i^{gsl}, as shown in the Add&Norm layer of Fig. 2(c), using skip-connections and layer normalization. The calculation method for the Add&Norm layer is as follows:

$$z_i^{gsl} = \text{LayerNorm}\left(z_i^{gsl} + h_i^0\right). \tag{15}$$

After conducting experiments, we observed that the hand-crafted connectivity yielded better results when the Add&Norm layer was not utilized. This could be attributed to the stability of the manually defined structure.

Then, z_i^{gsl} and z_i^{hcs} are concatenated together to be used for prediction, the overall pipeline can be seen in Fig. 2(c). In our experiments, we utilized MLP with the same structure as

(9) for prediction, where the specific number of MLP layers was determined as a hyperparameter. In this way, MPCFIN successfully models tabular data.

C. Complexity Analysis

The complexity of the MPCFIN model is influenced by three components: the cross-feature module, the multiplex GNN module, and the output module.

Feature Transformation Complexity: During the data processing, the model uses normalization and discretization functions to unify numerical and categorical features, including methods like LabelEncoder, One-Hot encoding, and embedding layers. These methods affect the model's performance:

- Normalization and discretization involve $O(N \cdot M)$ operations, where N is the number of samples and M is the number of features.
- Encoding adds to the preprocessing complexity, typically $O(N \cdot M)$ for LabelEncoder and $O(N \cdot M \cdot C)$ for One-Hot encoding, where C is the number of categories.

Cross-Feature Interaction Complexity: The model first initializes the column embeddings, then automatically learns the most relevant features for each attribute in the tabular dataset through similarity calculations to model feature interactions:

- Initializing embeddings for M features, each with an embedding dimension d, involves $O(M \cdot d)$ operations
- Caculating similarities for each pair of M features requires $O(M^2 \cdot d)$ operations
- For each feature, concatenating it with its most correlated features involves $O(M \cdot d)$ operations.

Graph Structure Learning Complexity: The model uses a multiplex graph structure to integrate handcrafted and automatically learned connections, enhancing the complexity of the graph representation. In the Graph Structure Learner (GSL), the model uses a metric-based approach to learn the graph structure, optimizing computational complexity and generating the adjacency matrix from the probability matrix:

- Constructing a hand-crafted graph involves $O(M)$ operations for connecting each node with the CLS node.
- In GSL, Learning the structure involves calculating similarities and generating the adjacency matrix, which is $O(M^2 \cdot d)$.
- In GNN, message passing for T iterations over M nodes, each with d-dimensional embeddings, involves $O(T \cdot M \cdot d)$ operations.

Outputs Module Complexity: This module involves the final prediction step using the concatenated embeddings from the CLS nodes of both graphs. The prediction process involves passing the concatenated embeddings through MLP layers. If there are L layers, each with d-dimensional input, the complexity is $O(L \cdot d^2)$.

Overall Complexity of MPCFIN: Combining the complexities of all modules, the overall complexity of the MPCFIN model can be approximated as:

$$O\left(N \cdot M + M^2 \cdot d + T \cdot M \cdot d + L \cdot d^2\right) \qquad (16)$$

TABLE I
The Statistical Information About 7 Datasets

Dataset	#Instances	#Num.	#Cat.	Task
Private Dataset	995	43	1	Multi-Class(4)
Gesture Phase	9873	32	0	Multi-Class(5)
Churn Modeling	10,000	9	1	Binary
California Housing	20,640	8	0	Regression
House 16H	22,784	16	0	Regression
Adult Income	32,561	6	8	Binary
Helena	65,196	27	0	Multi-Class(100)

#Num. indicates the number of numerical features, and #Cat. indicates the number of categorical features.

Given that M (number of features) and d (embedding dimension) are dataset-dependent and typically large, while T (iterations in GNN) and L (layers in MLP) are constants, the dominant term is $M^2 \cdot d$. The overall complexity can be simplified to $O(M^2 \cdot d)$.

V. Experiments

A. Experimental Setup

Datasets: We select a total of seven datasets, including one private dataset and six public datasets. The statistical information of these datasets is presented in Table I.

Private Dataset is obtained from our collaborative project with Wuhan Aier Eye Hospital Group Co. Ltd and consists of classification and diagnostic data for four types of eye diseases including one normal label and three diseases labels. This dataset comprises a total of 44 features, including one categorical feature.

Gesture Phase [59] consists of features extracted from seven videos containing hand gestures. It includes a total of 32 attribute features, most of which describe the coordinates and size of finger joints. The ultimate goal is to classify the gestures into one of five categories.

Churn Modeling [60] is sourced from Kaggle and was created by Shubham Kumar. Originally, the dataset consisted of 13 features. However, since the first three features were row numbers, user IDs, and user names, we did not include them. The final dataset was used to predict which customers will likely churn from the organization.

California housing [61] is a classic benchmark that includes data on houses found in a certain region of California, along with some aggregated statistics based on the 1990 census data. All the features in this dataset are numerical and are used for regression tasks to predict housing prices at that time.

House 16H is a dataset sourced from OpenML, meticulously crafted based on data provided by the American Census Bureau. It encompasses a total of 16 features, which are employed to forecast the median house price in a given location.

Adult Income [62] leverages population census data to predict whether an individual's income exceeds 50K\$per year, commonly referred to as the Census Income. This comprehensive dataset encompasses numerous personal attributes, such as education level, duration of education, and weekly working hours, among others.

TABLE II
THE RESULTS OF PERFORMANCE COMPARISON ON PUBLIC DATASETS

Models	Ges.Pha.		Chu.Mod.		Adu.Inc.		Hel.		Cal.Hou.	Hou.16H.
	ACC↑	AUC↑	ACC↑	AUC↑	ACC↑	AUC↑	ACC↑	AUC↑	MSE↓	MSE↓
SVM	55.65	80.05	85.61	82.70	84.88	89.20	35.73	87.50	0.354	0.483
MLP	61.42	80.69	86.14	84.68	85.41	90.80	36.88	88.66	0.275	0.341
NODE [40]	65.91	85.51	85.98	85.43	85.57	91.11	35.41	88.16	0.276	0.348
TabTransformer [44]	54.63	71.32	83.63	86.04	84.31	86.06	27.85	59.23	0.445	0.451
SAINT [12]	56.76	81.74	86.65	86.79	86.65	91.80	33.46	87.45	0.244	0.355
INCE [13]	66.05	85.67	86.87	86.75	86.81	91.89	35.71	87.82	0.216	0.332
T2G-Former [28]	65.77	**86.96**	87.06	86.45	86.46	91.76	**39.11**	**89.74**	0.265	0.333
MPCFIN	**66.28**	86.08	**87.33**	**87.24**	**87.02**	**91.98**	38.29	88.94	**0.207**	**0.324**

The best results are marked in bold and the second best results are underlined.

Helena [63] originates from the AutoML Challenge and, despite its modest 27 features, presents the task of classifying these samples into 100 distinct categories, with the added complexity of potential class imbalance.

Baselines: In Section II, we have categorized the existing methods into two main groups: hybrid models and Transformer-based models. Beyond these groups, we also consider models based on Graph Neural Networks (GNNs) and conventional machine learning techniques. Specifically, for machine learning techniques, we have conducted comparisons using SVM and MLP, which are widely recognized as foundational techniques in numerous studies. For hybrid models, we have examined the NODE [40] model. Besides, the Transformer-based models (TabTransformer [44], SAINT [12]), GNNs-based model (INCE [13]) and another multiplex graph network called T2G-Former [28] are included for comparison.

Implementation Details: All models undergo standardized preprocessing as described in (3), with missing or uncertain values imputed as zeros. The dataset is divided into 20% for testing and 80% for training. Hyperparameter fine-tuning is iteratively performed using the Optuna library [64] on each training set and model pair, with the training set further divided into a 10% validation set. If optimal hyperparameter configurations are provided in the original papers of the models, further tuning is omitted.

In the case of the Interaction Network (IN), the main hyperparameters include the dimension of the hidden vector, the number of IN layers, layers in feature mapper MLP, and layers in prediction MLP. Our experiments tested hidden vector sizes of 16, 32, 64, and 128. The results show that 32 and 64 are the most effective for most datasets. We also varied the number of IN layers (1-4) and MLP layers (1-4) to determine their impact. These four hyperparameters are automatically tuned using the Optuna library, eliminating the need for manual tuning. If necessary, manual tuning only requires adjusting 16 sets of parameters based on hidden vector sizes of 32 and 64.

We conduct all experiments on NVIDIA RTX 3090 during training using PyTorch with Python 3.8. At the same time, MPCFIN utilizes the Adam optimizer with a learning rate of 0.001 and a batch size of 256, trained for 200 epochs with early stopping implemented within 10 steps. Other models are configured as closely as possible for consistency, and the final experimental results are obtained by averaging three independent trials. To accurately replicate the performance of each model, we use the scikit-learn package for SVM and the official code provided by authors for other models. Additionally, performance is evaluated using MSE metric for regression tasks and accuracy/AUC (the area under the receiver operating characteristic curve) metric for classification tasks.

B. Experimental Results

Comparison With Counterparts: The average results of 3 trials for the model comparison experiments on the six public datasets are shown in Table II. In the DNNs methods, we highlight the best-performing results in bold and underline the second-best results. The results show that MPCFIN outperforms all other DNN models on five public datasets and achieves the second-best performance on the remaining dataset. Furthermore, we observe that methods based on GNNs generally outperform other DNNs methods, indirectly demonstrating the effectiveness of GNNs in modeling tabular data. However, the results demonstrate that the stability of the MPCFIN model is slightly inferior to models that use attention mechanisms. This is because the GSL module in MPCFIN learns the connected edges based on the input features at each training step, resulting in less stable edge generation with limited samples.

Additionally, the average performance comparison results on private dataset are shown in Table III. In the field of medicine, the assessment of false negative rate (FNR) and false positive rate (FPR) is crucial in medical diagnosis scenarios, going beyond mere accuracy. Hence, in Table III, our analysis extends beyond comparing accuracy (ACC) and area under the curve (AUC) alone. We delve into a comprehensive comparison by including FNR and FPR across the three disease labels present in the private dataset. This approach allows for a thorough evaluation of multiple models' performance on the private dataset. In Table III, the results reveal that MPCFIN emerges as the second-best model in terms of overall accuracy and area under the curve. However, upon closer examination of false negative rate and false positive rate, MPCFIN significantly outperforms the model with the highest overall accuracy. Notably, the false negative rate and false positive rate of MPCFIN rank among the top two models across all metrics. Thus, considering a comprehensive assessment of these evaluation indicators, MPCFIN demonstrates the most superior performance on the private dataset.

TABLE III
THE RESULTS OF PERFORMANCE COMPARISON ON PRIVATE DATASET WITH DISEASE DIAGNOSIS

Models	**ACC↑**	**AUC↑**	**FNR(1)↓**	**FNR(2)↓**	**FNR(3)↓**	**FPR(1)↓**	**FPR(2)↓**	**FPR(3)↓**
NODE [40]	89.17	95.94	34.62	23.33	<u>3.80</u>	4.02	3.53	2.48
TabTransformer [44]	89.67	96.08	**23.08**	<u>20.01</u>	5.06	5.17	3.17	**0.83**
SAINT [12]	88.50	95.18	31.58	28.13	6.15	**2.76**	<u>2.98</u>	3.70
INCE [13]	89.06	96.50	31.25	20.83	6.14	4.02	4.12	3.31
T2G-Former [28]	**91.50**	**98.53**	30.78	26.67	7.59	5.74	4.71	2.48
MPCFIN	<u>91.25</u>	<u>96.80</u>	<u>26.92</u>	**16.67**	**2.53**	<u>2.87</u>	**2.94**	<u>1.65</u>

The best results are marked in bold and the second best results are underlined. In addition to reporting ACC and AUC, we also provide the false negative rate (FNR) and false positive rate (FPR) of the private dataset on three disease labels.

TABLE IV
THE RESULTS OF DIFFERENT CROSS-FEATURE APPROACHES, DIFFERENT COMBINATIONS AND DIFFERENT GRAPH STRUCTURES

Methods	**Pri.Dat.**	**Chu.Mod.**	**Cal.Hou.**
	ACC↑	**ACC↑**	**MSE↓**
Different Cross-Feature Strategies			
Single Feature	88.67	86.75	0.215
Repeatable Sampling	88.50	86.22	0.223
Non-Repeatable Sampling	90.67	87.25	0.212
Adaptive Sampling (Ours)	91.25	87.33	0.207
Different Combinations			
SF & FCG	87.17	86.87	0.216
SF & MPG	88.67	86.75	0.215
CF & FCG	89.67	87.15	0.211
CF & MPG (Ours)	91.25	87.33	0.207
Different Graph Structures			
GSL	87.75	86.53	0.224
HCS	88.75	86.90	0.214
MPG (Ours)	91.25	87.33	0.207

SF: Single Feature. CF: Adaptive Cross-Feature. FCG: Fully Connected Graph. MPG: Multiplex Graph. GSL: Graph Structure Learner. HCS: Hand-Crafted Structure.

Ablation Experiments: To validate the effectiveness of the proposed modules in MPCFIN, we conduct ablation experiments. These experiments aim to assess the effectiveness of cross-feature modeling, the effectiveness of multiplex modeling, and the necessity of incorporating the Add&Norm layer. For convenience, we do not perform ablation experiments on all datasets. Instead, we select three datasets, namely California Housing, Churn Modeling, and a private dataset, for the ablation experiments.

Different Cross-feature Modeling Strategies: During the experiments, we investigate various strategies for cross-feature modeling in Table IV. These approaches encompass repeatable random sampling for cross-feature fusion where n features are randomly sampled with replacement for each feature fusion, non-repeatable random sampling for cross-feature fusion where n features are randomly sampled without replacement for each feature fusion, and the adaptive cross-feature sampling method proposed in our cross-feature embedding module. The first strategy, repeatable random sampling with replacement, involves randomly selecting k features from an infinite feature set to concatenate with each feature, forming cross-features. However, this method can lead to complete redundancy and unavoidable noise, especially in datasets with limited features. The second strategy, non-repeatable random sampling without replacement, divides a finite feature set into k non-repetitive combinations. Each feature within a combination is concatenated to form k sets of cross-features. This strategy avoids noise from redundant cross-features and explores hidden relationships between features. Nonetheless, its limitation lies in the fixed nature of the cross-feature combinations, which cannot be altered even if deeper connections exist among the combined features. Building on the first two strategies, we propose our strategy: learning a correlation matrix to guide each feature on which features to concatenate with. This approach ensures randomness and diversity in cross-feature combinations and autonomously learns the most suitable correlation matrix, enhancing the strategy's adaptability and effectiveness. Furthermore, we explore methods focusing on individual features to ensure comprehensive comparison. Clearly, cross-feature modeling methods perform better than those focusing on single feature, especially when not using repeatable sampling. The reason is that repeatable sampling can cause overlapping information between features, leading to unnecessary duplication.

Different Combinations: The impacts of employing cross-feature modeling and combining multiplex graphs are presented in the different combinations part of Table IV, where ”SF” means single feature, ”FCG” means fully connected graph, ”CF” means adaptive cross-feature, and ”MPG” means multiplex graphs. Since the multiplex graphs incorporate two different graph structures, we also separate them for individual comparisons with the module of cross-feature embedding. The results are shown in the different graph structures part of Table IV, where GSL and HCS are displayed in Fig. 2. In addition, we concatenate the vectors twice when using a single GSL or HCS structure to maintain the same dimensionality of the vectors used for prediction as in MPG.

Add&Norm Layer: As mentioned earlier, the Add&Norm layer can prevent gradient vanishing or exploding issues that may hinder model convergence during the learning process. However, careful exploration is required to determine where to add this layer and after which graph to add it. Therefore, we try to add the Add&Norm layer after both GSL and HCS separately, resulting in four different scenarios. Through experiments on the California Housing dataset, we found that adding the Add&Norm layer after the automatically learned graph produced better results, stabilizing the learning process. The experimental results are shown in Fig. 4, where the plus sign (+) represents the addition of the Add&Norm layer, while the minus sign (−) represents its absence.

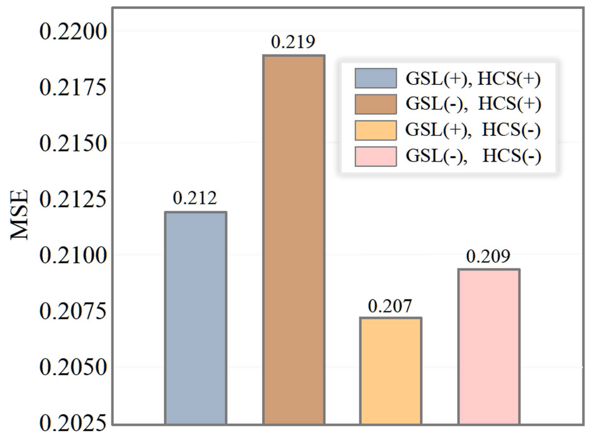


Fig. 4. The result on California Housing dataset of adding an Add&Norm layer after a different graph structure. The plus sign(+) represents the addition of the Add&Norm layer and the minus sign(-) represents its absence.

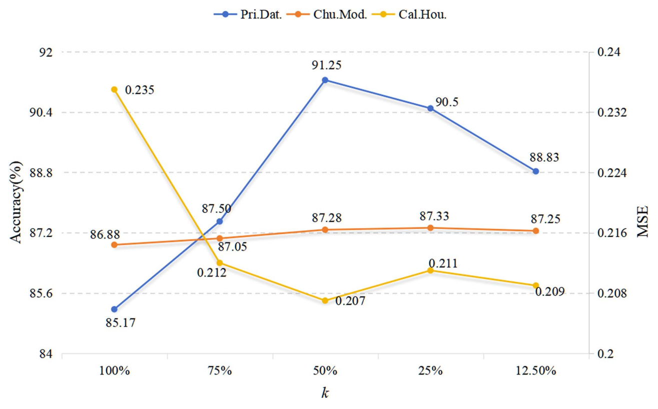


Fig. 5. The accuracies of different values of k on three datasets.

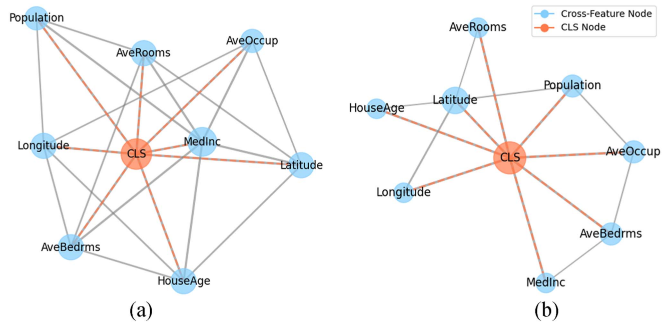


Fig. 6. Visualization of graph structure learned by Graph Structure Learner(GSL) on California Housing dataset with different parameters k. (a) k is set to 50%. (b) k is set to 25%.

TABLE V
THE COMPUTATIONAL COMPLEXITY OF DIFFERENT MODELS

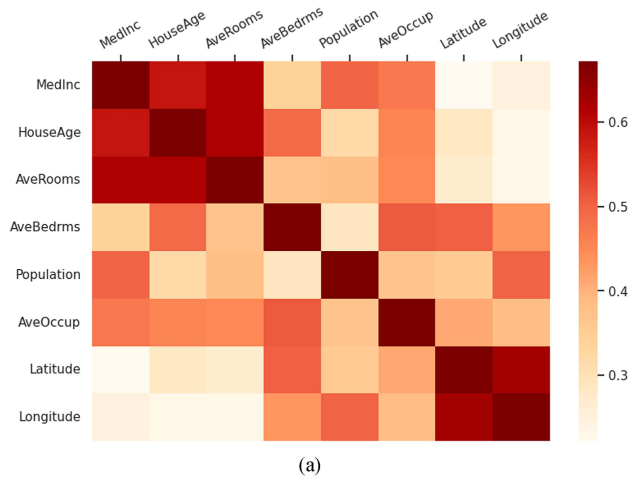
Models	Params	MACs
INCE [13]	44.844K	630.383M
T2G-Former [28]	1664.4K	30380.6M
MPCFIN	138.5K	3724M
MPCFIN-Cross Feature Module	4K	0.704M
MPCFIN-Multiplex GNN Module	91.246K	3721.7M
MPCFIN-Outputs Module	2.244K	0.557M

Params indicates the number of trainable parameters, and MACs indicates multiply accumulate operations.

Sensitivity of Parameter T in AFC: In the Adaptive Features Combiner (AFC), the threshold T is uniformly set to 0.5 after the probability score matrix passes through the activation function. This facilitates exploratory interactions between relevant features in order to minimize cross-feature redundancy. Furthermore, we have conducted additional experiments to analyze the sensitivity of hyperparameters on the threshold T. The results indicate that the impact of parameter T varies across different data sets. To ensure fairness and convenience, we have chosen to uniformly set T to 0.5. This decision enables the AFC to effectively balance the trade-off between capturing relevant feature interactions and minimizing unnecessary redundancy. The sensitivity analysis reveals that although performance varied across datasets, setting T to 0.5 provided a robust baseline, facilitating the exploration of feature interactions without overfitting to specific datasets.

Sensitivity of Parameter k & Visualization in GSL: In the Graph Structure Learner (GSL), one of the most crucial hyperparameters is k which governs the number of generated edges and consequently influences the model performance. In GSL, k is not a specific numerical value but rather a percentage. In Fig. 5, we investigate the impact of different values of k on the model performance, utilizing the same dataset as in the ablation experiments. From the experimental results, it is important to find a balance, avoiding excessively high or low values. For instance, if k is set to 25%, it indicates that each node is connected to 25% of the remaining total number of nodes to generate edges. As k approaches 100%, the graph becomes increasingly dense, resembling a fully connected graph. In addition, in Fig. 6, we visualize the graph structures learned by GSL on California Housing dataset with k values of 50% and 25%, respectively. To facilitate observation, each node is named after its corresponding feature prior to cross-feature fusion, except for the CLS node. The majority of nodes in the graph are connected to the CLS node and capture strong correlations between relevant features. This validates the rationality of the generated graph structure.

Computational Complexity of Different Models: To validate the complexity of the MPCFIN model, we have conducted experiments comparing its computational cost on the private dataset against other GNNs-based models. The results are summarized in Table V. It is evident that the computational complexity of GNNs-based models is significantly lower compared to attention-based model (T2G-Former). This can be attributed to the fact that GNNs are able to efficiently process and analyze graph-structured data, resulting in reduced computational requirements. Additionally, it is worth noting that the main source of computational complexity for MPCFIN lies in its multiplex GNN module, which introduces additional processing overhead due to its ability to handle multiple paths within a graph simultaneously.



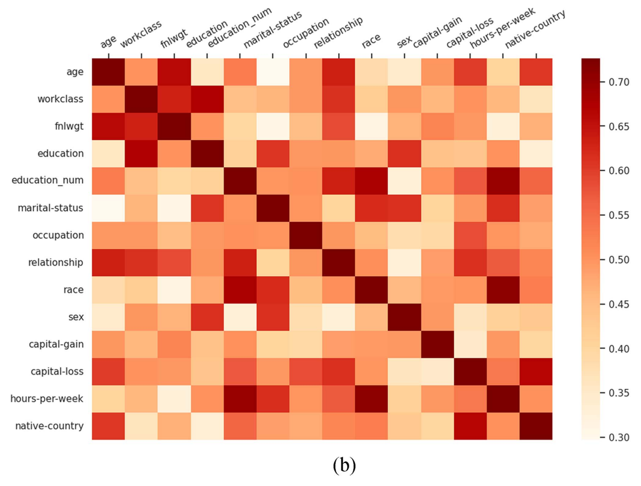


Fig. 7. Viasualization of the correlation adjacency matrices learned by the adaptive features combiner without undergoing any thresholding with parameter T. (a) California housing. (b) Adult income.

C. Interpretability of Adaptive Features Combiner

The adaptive features combiner automatically learns the most relevant features for each input feature, making it essential to investigate its interpretability. In Fig. 7, we visualize the correlation adjacency matrices learned by the Adaptive Features Combiner in two datasets, namely California housing and Adult income. Note that these matrices represent the correlations between features (as defined in (5)) without undergoing any thresholding with parameter T. In Fig. 7(a), we can observe that the median income (MedInc) in a region is closely correlated with the average house age (HouseAge) and the average number of rooms per dwelling (AveRooms). This correlation is reasonable in real life, as wealthier areas tend to have more and larger houses. Additionally, we can observe that the average number of occupants per household (AveOccup) is strongly correlated with both the average number of rooms per dwelling (AveRooms) and the average number of bedrooms per dwelling (AveBedrms). This correlation aligns with our daily experience. Furthermore, we can also notice that the population size (Population) is most correlated with the median income (MedInc) of the area, which is consistent with the idea that wealthier areas tend to have larger populations. Apart from these examples, there are other correlations that are also reasonable, such as Latitude-Longitude relationships. In Fig. 7(b), we can see that the working class is closely correlated with education level. This correlation is intuitive, as certain types of work are associated with specific educational backgrounds. Similar examples can be found in the heatmap, but we won't delve into them extensively here.

VI. Conclusion

This paper focuses on designing a graph neural network for modeling tabular data, aiming to address challenges pertaining to cross-feature information interaction and the connectivity of graph structure. Methodologically, we propose a Multiplex Cross-Feature Interaction Network (MPCFIN) that automatically learns the most relevant features for each feature and combines them to obtain cross-feature information. Then, it performs message passing along multiplex graph that combines the handcrafted and auto-learned structure. We empirically show that MPCFIN can efficiently handle the tabular modeling challenges and efficiently solve the cross-feature and connectivity issues faced by GNNs. The results of extensive experiments on various datasets substantiate that MPCFIN surpasses other deep neural network (DNN) models in terms of performance and exhibits remarkable interpretability through its cross-feature embedding module.

References

[1] Y. LeCun et al., "Handwritten digit recognition with a back-propagation network," in *Proc. Adv. Neural Inf. Process. Syst.*, 1989, pp. 396–404.
[2] J. L. Elman, "Finding structure in time," *Cogn. Sci.*, vol. 14, no. 2, pp. 179–211, 1990.
[3] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 6000–6010.
[4] J. Gu et al., "Recent advances in convolutional neural networks," *Pattern Recognit.*, vol. 77, pp. 354–377, 2018.
[5] Z. Niu, G. Zhong, and H. Yu, "A review on the attention mechanism of deep learning," *Neurocomputing*, vol. 452, pp. 48–62, 2021.
[6] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowl.-Based Syst.*, vol. 46, pp. 109–132, 2013.
[7] L. Wu, X. He, X. Wang, K. Zhang, and M. Wang, "A survey on accuracy-oriented neural recommendation: From collaborative filtering to information-rich recommendation," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 5, pp. 4425–4445, May 2023.
[8] Y. Liu, L. Chen, X. He, J. Peng, Z. Zheng, and J. Tang, "Modelling high-order social relations for item recommendation," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 9, pp. 4385–4397, Sep. 2022.
[9] J. Zhao, P. Zhao, L. Zhao, Y. Liu, V. S. Sheng, and X. Zhou, "Variational self-attention network for sequential recommendation," in *Proc. IEEE 37th Int. Conf. Data Eng.*, 2021, pp. 1559–1570.
[10] I. Kononenko, "Machine learning for medical diagnosis: History, state of the art and perspective," *Artif. Intell. Med.*, vol. 23, no. 1, pp. 89–109, 2001.
[11] R. J. Bolton and D. J. Hand, "Statistical fraud detection: A review," *Stat. Sci.*, vol. 17, no. 3, pp. 235–255, 2002.
[12] G. Somepalli, M. Goldblum, A. Schwarzschild, C. B. Bruss, and T. Goldstein, "SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training," 2021, *arXiv:2106.01342*. [Online]. Available: https://api.semanticscholar.org/CorpusID:235293989
[13] M. Villaizán-Vallelado, M. Salvatori, B. C. Martinez, and A. J. S. Esguevillas, "Graph neural network contextual embedding for deep learning on tabular data," *Neural Netw.*, vol. 173, 2024, Art. no. 106180.

[14] N. Frosst and G. E. Hinton, "Distilling a neural network into a soft decision tree," 2017, *arXiv: 1711.09784*. [Online]. Available: https://api.semanticscholar.org/CorpusID:3976789

[15] G. Ke, Z. Xu, J. Zhang, J. Bian, and T.-Y. Liu, "DeepGBM: A deep learning framework distilled by GBDT for online prediction tasks," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 384–394.

[16] S. Ivanov and L. Prokhorenkova, "Boost then convolve: Gradient boosting meets graph neural networks," 2021, *arXiv:2101.08543*. [Online]. Available: https://api.semanticscholar.org/CorpusID:231662243

[17] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, "Deep neural networks and tabular data: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 6, pp. 7499–7519, Jun. 2024.

[18] F. Feng, X. He, H. Zhang, and T.-S. Chua, "Cross-GCN: Enhancing graph convolutional network with *k*k-order feature interactions," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 1, pp. 225–236, Jan. 2023.

[19] S. Kong, W. Cheng, Y. Shen, and L. Huang, "AutoSrh: An embedding dimensionality search framework for tabular data prediction," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 7, pp. 6673–6686, Jul. 2023.

[20] C. Chen, M. Ye, M. Qi, and B. Du, "SketchTrans: Disentangled prototype learning with transformer for sketch-photo recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 5, pp. 2950–2964, May 2024.

[21] G. Li et al., "A lightweight and accurate spatial-temporal transformer for traffic forecasting," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 11, pp. 10 967–10 980, Nov. 2023.

[22] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, pp. 1189–1232, 2001.

[23] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 1, pp. 4–24, Jan. 2021.

[24] D. Berberidis and G. B. Giannakis, "Node embedding with adaptive similarities for scalable learning over graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 33, no. 2, pp. 637–650, Feb. 2021.

[25] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph CNN for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 1–12, 2019.

[26] P. Battaglia et al., "Interaction networks for learning about objects, relations and physics," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4509–4517.

[27] L. Telyatnikov and S. Scardapane, "EGG-GAE: Scalable graph neural networks for tabular data imputation," in *Proc. Int. Conf. Artif. Intell. Statist.*, PMLR, 2023, pp. 2661–2676.

[28] J. Yan, J. Chen, Y. Wu, D. Z. Chen, and J. Wu, "T2G-Former: Organizing tabular features into relation graphs promotes heterogeneous feature interaction," in *Proc. AAAI Conf. Artif. Intell.*, 2023, pp. 10 720–10 728.

[29] M. Ye, J. Shen, X. Zhang, P. C. Yuen, and S.-F. Chang, "Augmentation invariant and instance spreading feature for softmax embedding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 2, pp. 924–939, Feb. 2022.

[30] U. G. Mangai, S. Samanta, S. Das, and P. R. Chowdhury, "A survey of decision fusion and feature fusion strategies for pattern classification," *IETE Tech. Rev.*, vol. 27, no. 4, pp. 293–307, 2010.

[31] M. Ye, Z. Wu, C. Chen, and B. Du, "Channel augmentation for visible-infrared re-identification," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 46, no. 4, pp. 2299–2315, Apr. 2024.

[32] H. Liu, Y. Zhu, and Z. Wu, "Knowledge graph-based behavior denoising and preference learning for sequential recommendation," *IEEE Trans. Knowl. Data Eng.*, vol. 36, no. 6, pp. 2490–2503, Jun. 2024.

[33] L. Cosmo, A. Kazi, S.-A. Ahmadi, N. Navab, and M. Bronstein, "Latent-graph learning for disease prediction," in *Proc. 23rd Int. Conf. Med. Image Comput. Comput. Assist. Intervention*, Lima, Peru, Springer, Oct. 4-8, 2020, pp. 643–653.

[34] A. Kazi, L. Cosmo, S.-A. Ahmadi, N. Navab, and M. M. Bronstein, "Differentiable graph module (DGM) for graph convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 2, pp. 1606–1617, Feb. 2023.

[35] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2016, pp. 785–794.

[36] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "CatBoost: Unbiased boosting with categorical features," in *Proc. Neural Inf. Process. Syst.*, 2018, pp. 6639–6649.

[37] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Neural Inf. Process. Syst.*, 2017, pp. 3149–3157.

[38] Q. Zheng et al., "Deep tabular data modeling with dual-route structure-adaptive graph networks," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 9, pp. 9715–9727, Sep. 2023.

[39] A. Thawani, J. Pujara, P. A. Szekely, and F. Ilievski, "Representing numbers in NLP: A survey and a vision," 2021, *arXiv:2103.13136*. [Online]. Available: https://api.semanticscholar.org/CorpusID:232335764

[40] S. Popov, S. Morozov, and A. Babenko, "Neural oblivious decision ensembles for deep learning on tabular data," 2019, *arXiv: 1909.06312*. [Online]. Available: https://api.semanticscholar.org/CorpusID:202573030

[41] Y. Lou and M. Obukhov, "BDT: Gradient boosted decision tables for high accuracy and scoring efficiency," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 1893–1901.

[42] G. Ke, J. Zhang, Z. Xu, J. Bian, and T.-Y. Liu, "TabNN: A universal neural network solution for tabular data," 2019. [Online]. Available: https://openreview.net/forum?id=r1eJssCqY7

[43] S. Ö. Arik and T. Pfister, "TabNet: Attentive interpretable tabular learning," in *Proc. AAAI Conf. Artif. Intell.*, 2021, pp. 6679–6687.

[44] X. Huang, A. Khetan, M. Cvitkovic, and Z. S. Karnin, "Tab-Transformer: Tabular data modeling using contextual embeddings," 2020, *arXiv: 2012.06678*. [Online]. Available: https://api.semanticscholar.org/CorpusID:229156048

[45] X. Guo, Y. Quan, H. Zhao, Q. Yao, Y. Li, and W.-W. Tu, "TabGNN: Multiplex graph neural network for tabular data prediction," 2021, *arXiv:2108.09127*. [Online]. Available: https://api.semanticscholar.org/CorpusID:237259833

[46] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," in *Proc. Adv. Neural Inf. Process. Syst.*, 2021, pp. 18 932–18 943.

[47] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. North Amer. Chapter Assoc. Comput. Linguistics*, 2019, pp. 4171–4186.

[48] H. Guo, R. Tang, Y. Ye, Z. Li, and X. He, "DeepFM: A factorization-machine based neural network for CTR prediction," 2017, *arXiv: 1703.04247*. [Online]. Available: https://api.semanticscholar.org/CorpusID:970388

[49] R. Wang, B. Fu, G. Fu, and M. Wang, "Deep & cross network for ad click predictions," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 1–7.

[50] R. Wang et al., "DCN V2: Improved deep & cross network and practical lessons for web-scale learning to rank systems," in *Proc. Web Conf.*, 2021, pp. 1785–1797.

[51] Y. Zhu et al., "A survey on graph structure learning: Progress and opportunities," 2021, *arXiv:2103.03036*.

[52] Y. Chen, L. Wu, and M. Zaki, "Iterative deep graph learning for graph neural networks: Better and robust node embeddings," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 19 314–19 326.

[53] D. D. S. Pilco and A. R. Rivera, "Graph learning network: A structure learning algorithm," 2018, *arXiv: 1905.12665*. [Online]. Available: https://api.semanticscholar.org/CorpusID:86394934

[54] X. Gao, W. Hu, and Z. Guo, "Exploring structure-adaptive graph learning for robust semi-supervised classification," in *Proc. IEEE Int. Conf. Multimedia Expo*, 2020, pp. 1–6.

[55] W. Jin, Y. Ma, X. Liu, X. Tang, S. Wang, and J. Tang, "Graph structure learning for robust graph neural networks," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2020, pp. 66–74.

[56] Y. Zhang, S. Pal, M. Coates, and D. Ustebay, "Bayesian graph convolutional neural networks for semi-supervised classification," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 5829–5836.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2015, pp. 1026–1034.

[58] E. Jang, S. S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," 2016, *arXiv:1611.01144*. [Online]. Available: https://api.semanticscholar.org/CorpusID:2428314

[59] R. C. Madeo, C. A. Lima, and S. M. Peres, "Gesture unit segmentation using support vector machines: Segmenting gestures from rest positions," in *Proc. 28th Annu. ACM Symp. Appl. Comput.*, 2013, pp. 46–52.

[60] S. Kumar, "Churn modeling," Kaggle, 2020. [Online]. Available: https://www.kaggle.com/datasets/shubh0799/churn-modelling

[61] R. K. Pace and R. Barry, "Sparse spatial autoregressions," *Statist. Probability Lett.*, vol. 33, no. 3, pp. 291–297, 1997.

[62] B. Becker and R. Kohavi, "Adult," UCI Machine Learning Repository, 1996, doi: 10.24432/C5XW20.

[63] I. Guyon et al., "Analysis of the automl challenge series," *Automated Mach. Learn.*, vol. 177, pp. 177–219, 2019.

[64] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2019, pp. 2623–2631.