

國立陽明交通大學
數據科學與工程研究所
碩士論文

Institute of Data Science and Engineering
National Yang Ming Chiao Tung University
Master Thesis

針對表格學習之具目標條件控制的跨維度與跨編碼注
意力網路

XDEAT: Cross-Dimensional and Cross-Encoding
Attention with Target-aware Conditioning in Tabular
Learning

研 究 生：鄭志權 (Cheng, Chih-Chuan)

指導教授：曾意儒 (Tseng, Yi-Ju)

中華民國 一一四年七月

July 2025

針對表格學習之具目標條件控制的跨維度與跨編碼注意力網路

XDEAT: Cross-Dimensional and Cross-Encoding Attention with
Target-aware Conditioning in Tabular Learning

研 究 生： 鄭志權

Student： Chih-Chuan, Cheng

指導教授： 曾意儒 博士

Advisor： Dr. Yi-Ju Tseng



July 2025

Taiwan, Republic of China

中華民國 一一四年七月

誌 謝

時光飛逝，兩年的研究生活也接近尾聲。本論文能順利完成，最感謝的是我的指導教授 - 曾意儒教授兩年來的悉心指導，當中不僅在研究態度及方法上啟發良多，在為人處事方面更是受益匪淺。再者，感謝昆衛、璧如以及哲宇在兩年的研究生涯中彼此相互砥礪前行。最後，要感謝親愛的家人能夠給予支持與鼓勵，讓我可以無後顧之憂地朝向自己的理想前進。

鄭志權 於

國立陽明交通大學 數據科學與工程研究所

中華民國 一一四年七月

針對表格學習之具目標條件控制的跨維度與跨編碼注意力網路

學生：鄭志權

指導教授：曾意儒 博士

國立陽明交通大學 數據科學與工程研究所 碩士班

摘 要

本論文提出一個新框架 XDEAT (具目標條件控制的跨維度與跨編碼注意力網路)，專為表格數據的監督式學習設計。該框架的核心由雙路徑編碼器與多層注意力模組組成。雙路徑編碼器會將每個輸入特徵拆分成兩種形式：原始數值表徵，用來保留資料的原始特性；以及目標感知表徵，用來捕捉與預測目標相關的訊息。這兩種表徵可以同時保留完整的資料特徵，又凸顯與任務有關的重要資訊，並一同送入後續的注意力模組進行更深入的特徵交互學習。注意力模組包含兩個關鍵機制：其一為跨維度自注意力，用於捕捉同一路徑內特徵間的依存關係；其二為跨編碼自注意力，用於促進原始與目標感知表徵之間的雙向交互。基於五個公開數據集的實驗結果顯示，XDEAT 在效能上持續優於強基線模型，驗證了同時建模原始與目標感知特徵的有效性。

關鍵字：表格深度學習, 目標編碼, 自注意力機制, 變換器, 表示學習

XDEAT: Cross-Dimensional and Cross-Encoding Attention with Target-aware Conditioning in Tabular Learning

Student : Chih-Chuan, Cheng

Advisor: Dr. Yi-Ju Tseng

Institute of Data Science and Engineering
National Yang Ming Chiao Tung University

Abstract

A novel framework, XDEAT (Cross-Dimensional and Cross-Encoding Attention with Target-Aware Conditioning), is proposed for supervised learning on tabular data. At its core, XDEAT employs a dual-stream encoder that decomposes each input feature into two parallel representations: a raw value stream and a target-conditioned (label-aware) stream. These dual representations are then propagated through a hierarchical stack of attention-based modules. XDEAT integrates two key components: (i) cross-dimensional self-attention, which captures intra-view dependencies among features within each stream; (ii) cross-encoding self-attention, which enables bidirectional interaction between raw and target-aware representations. Performance improvements over strong baselines are consistently observed across multiple benchmark datasets, demonstrating the effectiveness of jointly modeling raw and target-aware views through attention-based mechanisms.

Keywords: Tabular Deep Learning, Target Encoding, Self-Attention, Transformer, Representation Learning.

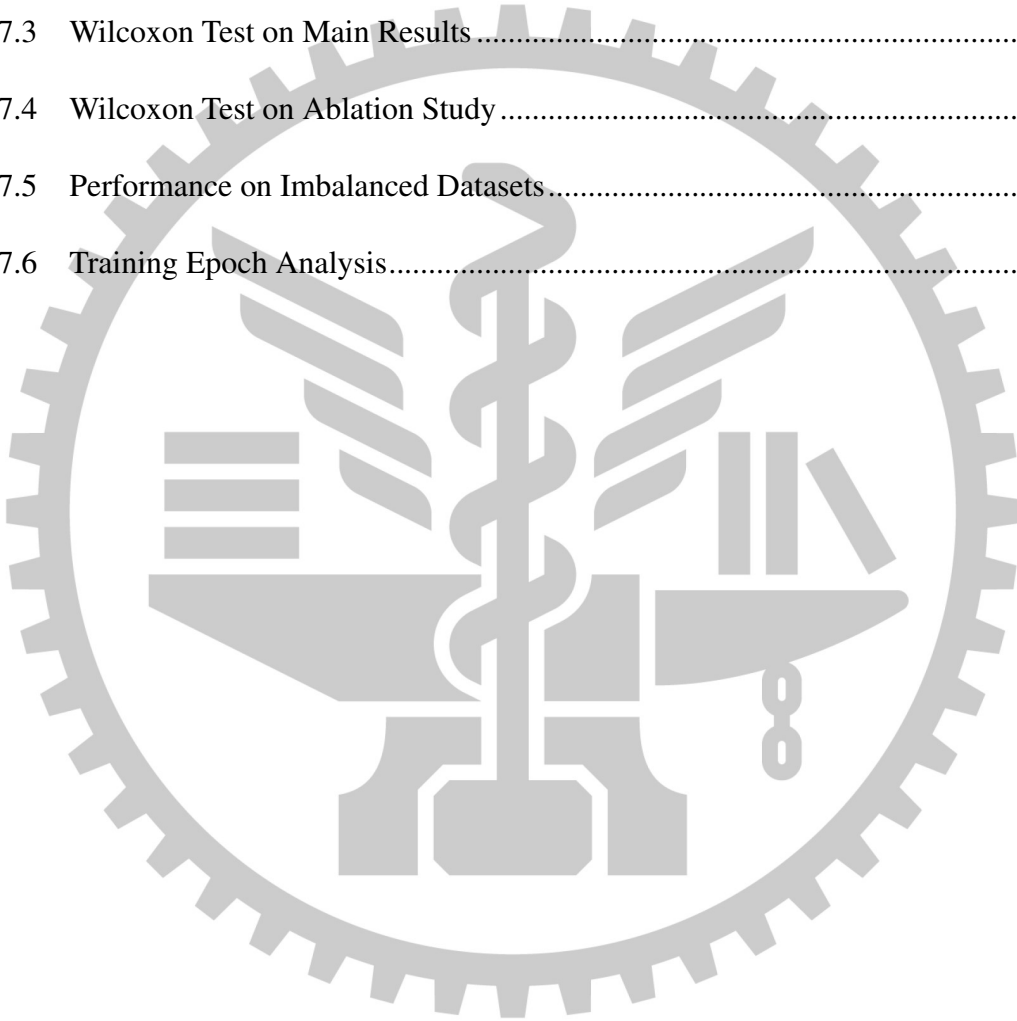
Contents

Chinese Abstract	i
English Abstract	ii
Contents	iii
List of Figures	vii
List of Tables	viii
1 Introduction	1
1.1 Background	1
1.1.1 Categorical Data Encoder	1
1.1.2 Machine Learning in Tabular Data	2
1.1.3 Deep Learning in Tabular Data	2
1.2 Research Motivation and Objectives	3
2 Related Work	4
2.1 Machine Learning with Tabular Data	4
2.2 Deep Learning with Tabular Data	4
2.2.1 Attention Based Models	5
2.2.2 Graph Based Models	5

2.2.3	LM-Based Models	6
2.2.4	Other Models	6
2.3	Deep Learning with Time Series Data.....	7
3	Methodology	8
3.1	Overview	8
3.2	Dual-Stream Encoding.....	9
3.2.1	M-Estimate Target Encoding for Classification Tasks.....	10
3.2.2	M-Estimate Target Encoding for Regression Tasks.....	15
3.2.3	Feature Tokenizer.....	16
3.3	Model Architecture	18
4	Experiment Settings.....	22
4.1	Datasets.....	22
4.1.1	California Housing.....	23
4.1.2	Adult	23
4.1.3	KDD Internet Usage	23
4.1.4	HIGGS Small.....	23
4.1.5	Jannis.....	24
4.2	Environment Setup.....	24
4.2.1	Optuna.....	24
4.2.2	Implementation Details.....	25
4.2.3	Evaluation Metrics	25

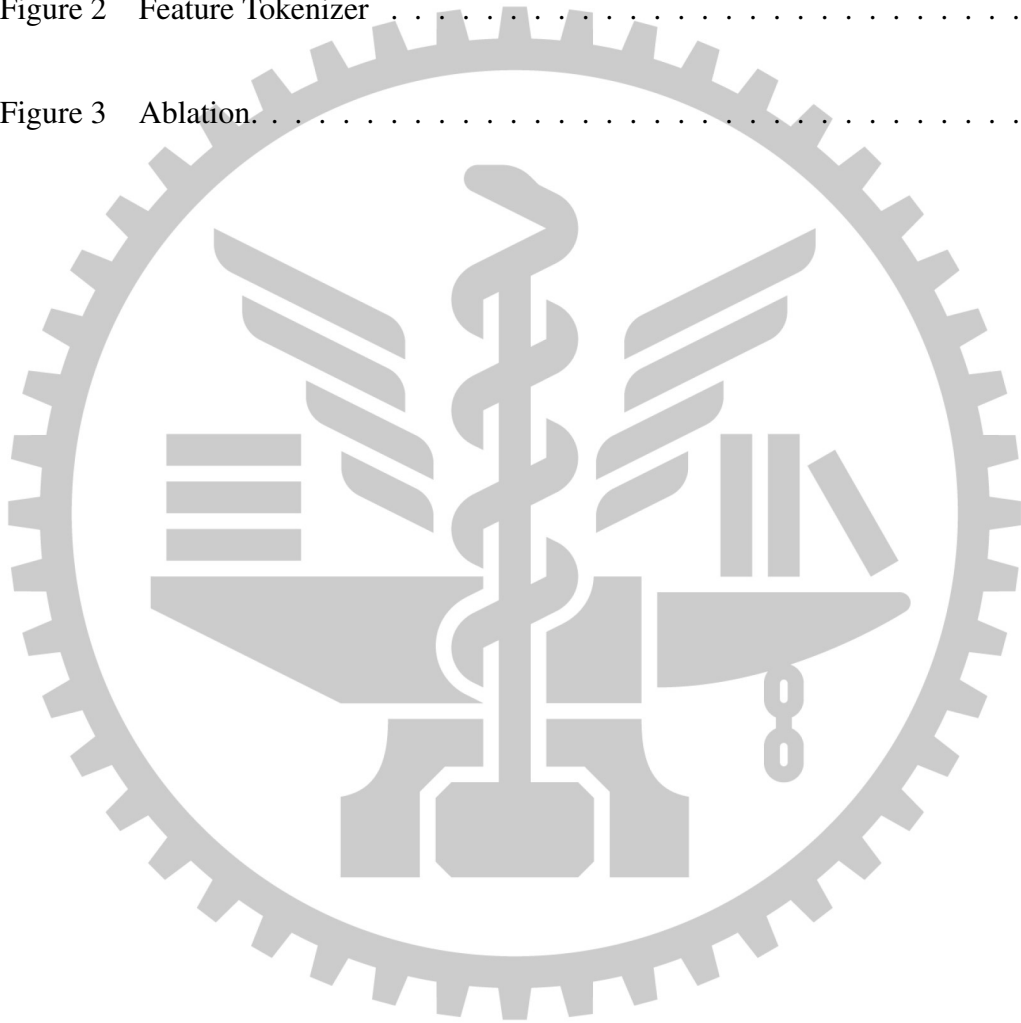
4.2.4	Preprocessing	25
4.2.5	Benchmark Model.....	26
5	Results.....	27
5.1	Main Results	27
5.2	Ablation Studies.....	28
6	Discussion and Limitations.....	34
6.1	Discussion.....	34
6.2	Limitations	35
7	Conclusions.....	36
	References.....	37
	Appendix.....	43
7.1	(Optuna) Model Hyper-Parameter Configuration	43
7.1.1	XGBoost	43
7.1.2	MLP	44
7.1.3	ResNet.....	44
7.1.4	DCN V2	45
7.1.5	NODE	46
7.1.6	AutoInt	46
7.1.7	FT-Transformer	46

7.2	Details of Target Encoders	47
7.2.1	Mean Encoder	47
7.2.2	S-Shrink Encoder	48
7.2.3	Beta Encoder	48
7.2.4	M-Estimate Encoder	49
7.3	Wilcoxon Test on Main Results	49
7.4	Wilcoxon Test on Ablation Study	51
7.5	Performance on Imbalanced Datasets	52
7.6	Training Epoch Analysis	53



List of Figures

Figure 1	XDEAT Architecture.	8
Figure 2	Feature Tokenizer	16
Figure 3	Ablation.	33



List of Tables

Table 1	Datasets description	22
Table 2	Comparison of performance across various benchmark models. Bold indicates the best performance across all models; <u>underline</u> indicates the second-best performance across all models.	28
Table 3	Performance of Various Target Encoding Methods with FT-Transformer. Bold indicates the best performance across all models; <u>underline</u> indicates the second-best performance across all models.	29
Table 4	Performance of Various Target Encoding Methods with XGBoost. Bold indicates the best performance across all models; <u>underline</u> indicates the second-best performance across all models.	30
Table 5	Comparison of performance and encoding time (averaged over 15 experiments, excluding model training) between non-binned and bin-based encoding strategies under M-Estimate Encoding for numerical features across three datasets: CA, HI, and JA.	32
Table 6	Performance comparison under different model components. Bold indicates the best performance across all models; <u>underline</u> indicates second-best performance across all models.	33
Table 7	XGBoost hyperparameter space for datasets.	44
Table 8	MLP hyperparameter space for datasets.	44

Table 9	ResNet hyperparameter space for datasets.	45
Table 10	DCN V2 hyperparameter space for datasets.	45
Table 11	AutoInt hyperparameter space for datasets.	46
Table 12	FT-Transformer hyperparameter space for datasets.	47
Table 13	Comparison of performance across various benchmark models. The best result and those not statistically different from it are all shown in bold . Values are reported as mean \pm standard deviation.	50
Table 14	Performance comparison under different model components. The best result and those not statistically different from it are all shown in bold . Values are reported as mean \pm standard deviation.	51
Table 15	More performance metrics (AUROC, AUPRC, F1 Score, Recall, and Pre- cision) across different imbalanced datasets (AD, KDD, and JA). Values are reported as mean \pm standard deviation.	52
Table 16	Average number of training epochs across datasets, determined by the early stopping mechanism (maximum capped at 200 epochs).	54

Chapter 1. Introduction

1.1 Background

In real-world applications, the most prevalent form of data is tabular data [1], where each sample (row) shares a consistent set of features (columns). Tabular data is widely used across diverse domains, including medicine, finance, transportation [2] [3].

1.1.1 Categorical Data Encoder

Tabular data comprises numerical and categorical features. Among these, categorical variables are a defining characteristic that poses unique challenges. Since most machine learning and deep learning models are designed to operate on numerical inputs, effectively encoding categorical features into a suitable numerical representation is essential [4].

Below is an overview of commonly used categorical encoding methodologies:

- **One-Hot Encoding (OHE):** Converts each category into a binary vector. Although widely used, it may lead to high-dimensional sparse representations [5].
- **Ordinal Encoding:** Assigns a unique integer to each category. This method is simple but may impose unintended ordinal relationships between categories [6].
- **Target Encoding:** Replaces each category with a statistic of the target variable (e.g., mean target value per category). However, with high-cardinality features and imbalanced categories, the estimate may be highly unreliable [7].

- **Regularized Target Encoding:** Introduces smoothing techniques to balance the global mean and category-specific statistics, improving generalization on rare categories [7].
- **Hash Encoding (Feature Hashing):** Applies a hash function to map categories into a fixed number of bins. It is memory-efficient for high-cardinality features, but introduces a risk of hash collisions [8], where distinct categories may be mapped to the same bin.
- **Binary Encoding:** Converts categories into a binary number and splits the bits into separate columns. It offers a trade-off between dimensionality and information preservation.

Several studies [9] [10] concluded that target encoding and its variants are competitive and effective methodologies for encoding categorical variables.

1.1.2 Machine Learning in Tabular Data

Machine learning has long been the dominant approach for modeling tabular data due to its superior performance [3], interpretability, and effectiveness on structured inputs. Models such as Gradient Boosted Decision Trees (GBDT) [11], and Random Forests [12] have demonstrated strong performance across various datasets [3] [13][14].

1.1.3 Deep Learning in Tabular Data

Applying deep learning to tabular data presents unique challenges. Unlike images or text, tabular features lack inherent spatial or sequential structure, and each feature may differ significantly in semantics, scale, and distribution. Early attempts using plain fully connected neural networks, such as multi-layer perceptrons (MLPs), often underperformed compared to tree-based ensemble methods on tabular benchmarks [2]. In recent years, however, a variety of specialized neural network architectures have been proposed to close this performance gap [2].

Nevertheless, many studies continue to report that deep learning methods do not consistently outperform traditional approaches across all tabular datasets. Notably, when the underlying data distribution is more regular—i.e., with fewer outliers and less heavy-tailed behavior—deep learning models tend to perform better, highlighting the importance of dataset characteristics in determining the most suitable modeling approach [13].

1.2 Research Motivation and Objectives

Traditional encoding methods like one-hot encoding can lead to high-dimensional, sparse representations that are not well-suited for neural networks. In contrast, target encoding provides a more compact and informative alternative by mapping each category to a statistic derived from the target variable, capturing category-specific predictive patterns. We aim to leverage the strengths of target encoding as a feature transformation strategy to improve deep learning performance on tabular data. The objective of this study is to design and evaluate a deep learning framework that integrates target-encoded inputs to enhance representation learning, model generalization, and predictive accuracy on real-world tabular datasets.

Chapter 2. Related Work

2.1 Machine Learning with Tabular Data

Gradient Boosting Decision Trees (GBDT), including XGBoost [15], LightGBM [16], and CatBoost [17], are widely used for tabular data due to their ability to handle mixed feature types, missing values, and non-linear patterns. These models offer strong predictive performance and remain the default choice in many structured data applications. In particular, CatBoost [17] demonstrates state-of-the-art performance across 9 benchmark datasets, consistently outperforming both XGBoost [15] and LightGBM [16] in terms of accuracy and stability.

However, most GBDT models rely on univariate splits, meaning each decision node considers only a single feature. This limitation may miss complex feature interactions, reducing their ability to capture intricate dependencies among features [2]. Additionally, GBDTs often exhibit bias toward majority classes when dealing with imbalanced datasets, requiring external techniques such as SMOTE or cost-sensitive learning to mitigate class imbalance issues [18].

2.2 Deep Learning with Tabular Data

Early deep learning approaches for tabular data commonly adopt multilayer perceptrons (MLPs), which utilize stacked fully connected layers to learn representations directly from raw input features. ResNet-inspired architectures introduce residual connections to facilitate optimization and enable deeper networks [19]. While both serve as strong baselines, they are

susceptible to overfitting in low-data settings and often require extensive hyperparameter tuning. To better understand recent advancements, we categorize deep tabular models into four groups: Attention-Based, Graph-Based, LM-Based, and Others.

2.2.1 Attention Based Models

Attention-based methods aim to model complex feature interactions more effectively. AutoInt [20] employs self-attention to automatically capture high-order interactions among features, reducing the reliance on manual feature engineering. FT-Transformer [21] extends Transformer architectures to tabular data by applying attention mechanisms to tokenized numerical and categorical features, demonstrating strong performance across various benchmarks. Although these models offer improved interpretability and expressiveness, they can be computationally expensive due to the quadratic complexity of self-attention. Recent studies [22] have introduced efficient attention variants to reduce computational cost, making Transformers more practical for large-scale tabular learning. However, the lack of standardized datasets and pipelines still limits fair comparison and slows progress in architecture selection.

2.2.2 Graph Based Models

Graph-based approaches aim to capture the complex relationships between features and instances by transforming tabular data into graph structures. GNN4TDL [23] represents a broader research direction that utilizes Graph Neural Networks (GNNs) for learning from tabular data. It involves designing suitable graph constructions, such as instance graphs, feature graphs, or bipartite graphs, and applying message passing to enhance feature interaction modeling. However, these methods often suffer from scalability limitations and high computational costs due to graph construction and propagation. A related approach, GANDALF [24], integrates dif-

ferentiable decision trees with neural networks and graph regularization, aiming to combine interpretability with learnable representations. While promising, both strategies require careful graph design and may not generalize well across heterogeneous tabular tasks.

2.2.3 LM-Based Models

A growing trend in tabular learning is to leverage large-scale pretrained language models by transforming tabular data into a text-like format. PTab [25] converts each row into a sentence of feature – value pairs and fine-tunes a language model (e.g., BERT) using masked language modeling and supervised objectives. It performs well when feature names and values carry semantic meaning aligned with the pre-trained model vocabulary. In contrast, TabLLM [26] adopts a meta-learning approach that enables generalization to new tabular tasks in a zero-shot or few-shot setting, eliminating the need for task-specific retraining or extensive fine-tuning. While PTab focuses on adapting to specific datasets via fine-tuning, TabLLM aims for task generalization through instruction tuning and in-context learning. Despite their strong performance, both approaches are computationally intensive and require careful engineering of the input text format, limiting their practicality in low-resource settings.

2.2.4 Other Models

Other models, such as DCN-V2 [27] and NODE [28], take alternative approaches to modeling feature interactions. DCN-V2 combines deep neural networks with explicit cross networks to capture bounded-degree feature combinations effectively. NODE (Neural Oblivious Decision Ensembles) integrates differentiable tree-like structures into deep learning, mimicking the behavior of decision tree ensembles while supporting end-to-end optimization. Although these models introduce useful inductive biases and offer improved interpretability, their performance

generally lags behind attention-based methods on complex tabular benchmarks [21].

2.3 Deep Learning with Time Series Data

Numerous studies have explored multivariate time series forecasting, with Crossformer [29] being a prominent example that addresses long-term forecasting by capturing both temporal and feature-wise dependencies through a dual-path attention mechanism. While Crossformer explicitly models temporal dynamics, our approach adopts a different perspective. Specifically, we propose a model architecture in which one path integrates both a target-encoded view, emphasizing task-specific relevance, and a raw-value view, preserving the original feature representations. Furthermore, similar to Crossformer [29], a second path is designed to capture inter-sample feature dependencies. This dual-view encoding enables the model to learn richer and more interpretable representations, thereby improving performance on downstream tasks. Additionally, the cross-dimension module in Crossformer employs fully connected layers and attention mechanisms, which, while reducing computation costs, may introduce noise—particularly in high-dimensional settings [29]. This issue is especially relevant to tabular data, where many features are common. Developing a more robust cross-dimension module is therefore an important future direction in this research.

Chapter 3. Methodology

3.1 Overview

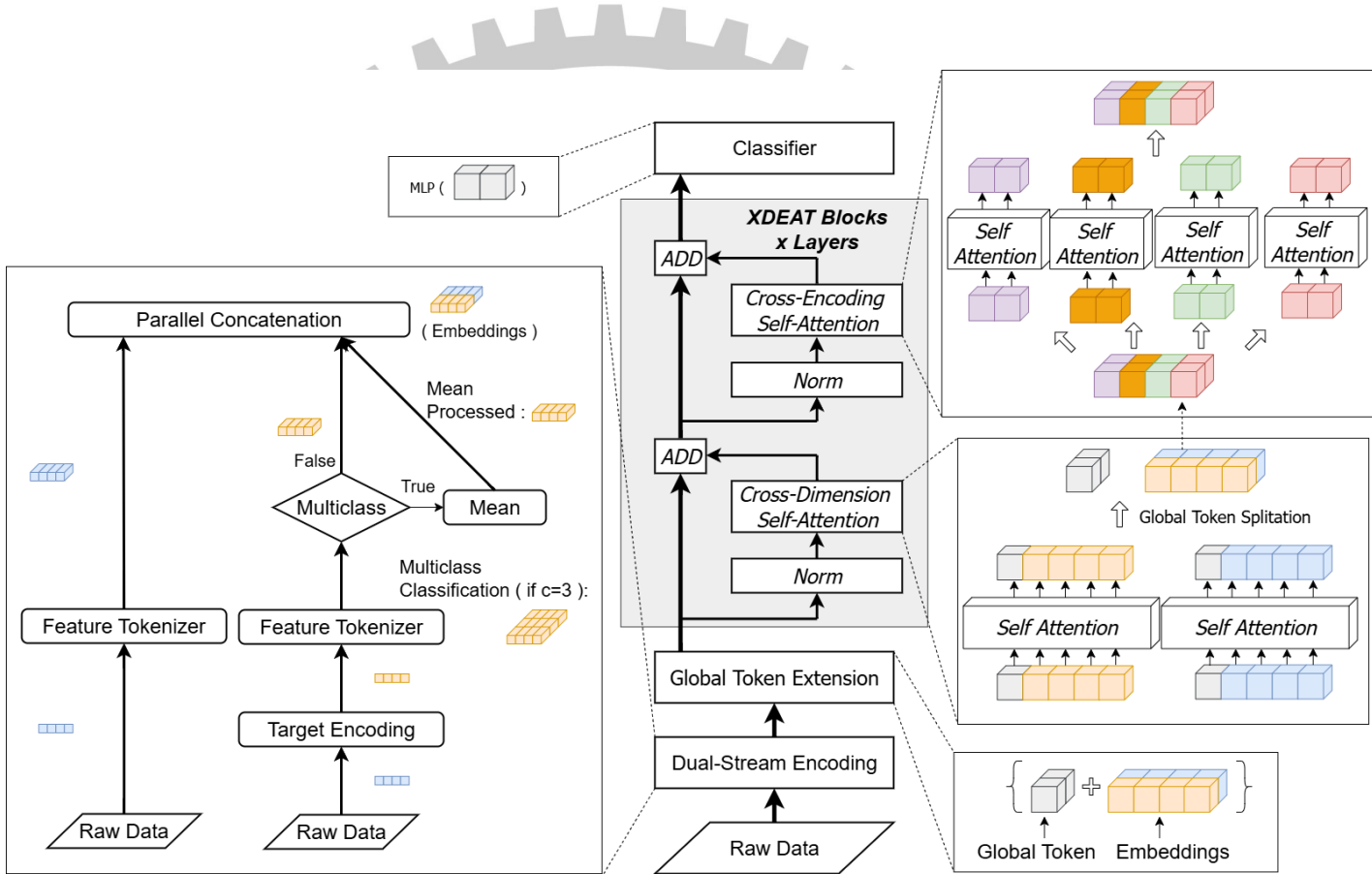


Figure 1: XDEAT Architecture.

Figure 1 presents the overall pipeline of our proposed architecture, XDEAT. The process begins with raw tabular data being input into the Dual-Stream Encoding module. Within this module, every feature is not only kept as is, but is also passed through a target encoding to generate a target-aware counterpart. This results in two simultaneous perspectives of the data: the original raw and the target-encoded views.

Then both views are individually processed by a shared feature tokenizer, which transforms each feature into a sequence of embeddings. To further enhance the ability to capture global context, learnable randomized global tokens are appended to the token sequences of both views in the Global Token Extension module.

The embeddings are fed into the XDEAT block, which first applies cross-dimensional self-attention within each view to model intra-sample dependencies. This is followed by a cross-encoding attention mechanism that allows the model to exchange information between the raw and target-encoded representations, enabling richer and more discriminative feature learning.

3.2 Dual-Stream Encoding

Encoding strategies are crucial for enabling machine learning models to effectively capture meaningful patterns from both categorical and numerical features. XDEAT adopts target-aware encoding schemes that incorporate information from the target variable in the Dual-Stream Encoding module. Specifically, motivated by the findings of Zhu et al. [30], M-Estimate Encoder is used for both regression and classification tasks, due to its ability to balance informativeness and time-effectiveness. To cover most of the prediction tasks, this work considers numerical and categorical features across three prediction tasks: binary classification, multi-class classification, and regression. M-Estimate encoding is applied accordingly. **Algorithm 1** details the encoding procedure for categorical features under binary classification, while **Algorithm 2** covers the corresponding approach for bin-based numerical features. Extensions to regression and multi-class classification are discussed in the later sections.

3.2.1 M-Estimate Target Encoding for Classification Tasks

The M-Estimate Target Encoder is a smooth target encoding technique that combines the category-specific mean of the target variable with the global mean, controlled by a smoothing parameter m . It helps prevent overfitting when the number of samples in some categories is small [7].

$$\hat{y}_c = \frac{n_c \cdot \bar{y}_c + m \cdot \bar{y}}{n_c + m}$$

Where:

- \hat{y}_c : Encoded value for category c
- n_c : Number of samples in category c
- \bar{y}_c : Mean target value for category c
- \bar{y} : Global mean of the target
- m : Smoothing parameter

Categorical M-Estimate Encoding (Binary Classification)

To avoid overfitting, combine the category mean and the global mean using a smoothing parameter m , typically set at 1% of the size of the training set.

Algorithm 1 M-Estimate Encoding for Categorical Features

Input: Training dataset D with categorical feature x and target y

- 1: Compute the global target mean: $\bar{y} = \text{mean}(y)$
 - 2: Set smoothing parameter: $m = 0.01 \times |D|$
 - 3: **Step 1: Compute smoothed target values for each category**
 - 4: **for** each category c in x **do**
 - 5: Count of samples in category: $n_c = \text{count}(x = c)$
 - 6: Mean target value for category: $\bar{y}_c = \text{mean}(y \text{ where } x = c)$
 - 7: Smoothed value:
$$\hat{y}_c = \frac{n_c \cdot \bar{y}_c + m \cdot \bar{y}}{n_c + m}$$
 - 8: Store result: $\text{Encoding}[c] \leftarrow \hat{y}_c$
 - 9: **end for**
 - 10: **Step 2: Apply encoding to all samples**
 - 11: **for** each sample in D **do**
 - 12: Replace category value x_i with $\text{Encoding}[x_i]$;
 - 13: if unseen, use global mean \bar{y}
 - 14: **end for**
-

Bin-Based Numerical M-Estimate Target Encoding (Binary Classification)

Numerical Estimate Encoding extends M-Estimate smoothing to numerical features by applying binning and local neighborhood aggregation. It first scales the numerical features to $[0, 1]$, then assigns each value to a bin starting from 0, using a fixed bin size (default = 0.001), resulting in bins such as $[0, 0.001)$, $[0.001, 0.002)$, \dots , $[0.999, 1.000]$. For each bin, the method computes a smoothed estimate of the target by aggregating values from nearby bins using M-

smoothing. The smoothing parameter m is set to 1% of the size of the training dataset.

This method defines the neighborhood of each bin by using the standard deviation of all bin indices, indicating the spread of the scaled numerical feature. This standard deviation determines how many adjacent bins on each side are included in the smoothing process. This strategy dynamically adapts the neighborhood size based on the feature distribution and ensures consistent smoothing across differently scaled variables.

To enhance efficiency, a bin-based approach is applied, thus avoiding finding neighbors for each individual numerical value, which would require processing all pairwise comparisons over the entire training set. Specifically, if neighbors were computed for each sample individually, the computational complexity would grow linearly with the size of the training dataset. In contrast, by discretizing the normalized feature range $[0, 1]$ into fixed-width bins (e.g., bin size = 0.001), the number of required neighborhood computations is reduced to approximately $1/\text{bin size}$. This drastically reduces the number of smoothing operations from $\mathcal{O}(N)$ to approximately $\mathcal{O}(1/\text{bin size})$, where N is the number of training samples. As a result, this bin-based strategy yields significant computational savings without sacrificing encoding quality.

Although target encoding is traditionally used for categorical variables, this method can also extract meaningful information from numerical features.

Algorithm 2 M-Estimate Encoding for Bin-Based Numerical Features

Input: Dataset D with numerical feature x and binary target y

- 1: Normalize feature values to $[0, 1]$: $x' = \text{MinMaxScaler}(x)$
- 2: Set bin size $s = 0.001$, smoothing factor $m = 0.01 \times |D|$
- 3: **Step 1: Group target values by bins**
- 4: **for** each data point $(x'_i, y_i) \in D$ **do**
- 5: Compute bin index: $b_i = \lfloor x'_i / s \rfloor$
- 6: Add y_i to list of targets in bin b_i
- 7: **end for**
- 8: **Step 2: Compute smoothed target for each bin**
- 9: Compute global target mean: $\bar{y} = \text{mean}(y)$
- 10: Determine neighborhood width r using standard deviation of bin indices
- 11: **for** each bin b with data **do**
- 12: Gather all target values from neighboring bins $[b - r, b + r]$
- 13: Let n_b be the number of collected values and \bar{y}_b their mean
- 14: Compute smoothed estimate:
$$\hat{y}_b = \frac{n_b \cdot \bar{y}_b + m \cdot \bar{y}}{n_b + m}$$
- 15: **end for**
- 16: **Step 3: Encode each sample**
- 17: **for** each data point $x'_i \in D$ **do**
- 18: Compute bin index: $b_i = \lfloor x'_i / s \rfloor$
- 19: Assign encoded value using the smoothed value of bin b_i ;
- 20: if not available, use the closest known bin or a weighted average of nearby bins
- 21: **end for**

Categorical M-Estimate Target Encoding (Multi Classification)

In the multi-class setting, M-Estimate encoding for categorical features is extended by calculating a separate smoothed estimate for each class. Given a categorical feature x and a multi-class target $y \in \{1, \dots, C\}$, the encoding produces a vector of length C for each category.

- For each class $c \in \{1, \dots, C\}$, we compute:

$$\hat{y}_{j,c} = \frac{n_{j,c} \cdot \bar{y}_{j,c} + m \cdot \bar{y}_c}{n_{j,c} + m}$$

where $n_{j,c}$ is the number of samples with feature value $x = j$ and target class c , $\bar{y}_{j,c}$ is the empirical probability of class c within category j , and \bar{y}_c is the global prior probability of class c across the dataset.

- The final encoding for category j is a vector:

$$\text{Encoding}[j] = [\hat{y}_{j,1}, \hat{y}_{j,2}, \dots, \hat{y}_{j,C}]$$

Bin-Based Numerical M-Estimate Target Encoding (Multi-Classification)

The multi-class of the bin-based numerical M-Estimate encoding follows the same binning strategy as in the binary classification case. Each numerical feature is first normalized and discretized into bins. As in the binary case, the target statistics for each bin are computed using not only the samples within the bin but also those from neighboring bins within one standard deviation of the bin index, which helps stabilize the estimates, especially in sparse regions.

The key distinction in the multi-class setting is that separate smoothed estimates must be computed for each class, resulting in a multi-dimensional encoding vector for each bin.

- For each class $c \in \{1, \dots, C\}$, here is the M-Estimate Encoding formula:

$$\hat{y}_{b,c} = \frac{n_{b,c} \cdot \bar{y}_{b,c} + m \cdot \bar{y}_c}{n_{b,c} + m}$$

where $n_{b,c}$ is the number of samples in bin b with class label c , $\bar{y}_{b,c}$ is the local proportion of class c in the neighborhood of bin b , and \bar{y}_c is the global class prior of class c .

- The final encoding for each bin b is a vector:

$$\text{Encoding}[b] = [\hat{y}_{b,1}, \hat{y}_{b,2}, \dots, \hat{y}_{b,C}]$$

3.2.2 M-Estimate Target Encoding for Regression Tasks

The core idea of M-Estimate encoding for regression task remains largely consistent with binary classification tasks. In the classification setting, the target values are binary (e.g., 0 or 1), and the encoding for each category or bin is computed based on the frequency of positive labels. In contrast, for regression, the target is set as a continuous numerical value, and the mean of those values is computed, rather than the frequency.

Categorical M-Estimate Encoding (Regression)

For each category, I compute the average of the target values associated with that category and then apply the same M-smoothing strategy used in the classification setting:

$$\hat{y}_c = \frac{n_c \cdot \bar{y}_c + m \cdot \bar{y}}{n_c + m}$$

Here, \bar{y}_c represents the mean target value for category c , \bar{y} denotes the global mean of all target values, and m is the smoothing parameter that controls regularization toward the global mean.

Bin-Based Numerical M-Estimate Encoding (Regression)

The feature is first normalized to $[0, 1]$, then discretized into fixed-width bins. For each bin, I collect the target values, compute their mean, and apply M-smoothing. The neighborhood of each bin is determined using the standard deviation of bin indices (or alternative dispersion measures), allowing for smoothed encoding that adapts to the local structure of the data.

In both cases, this encoding enables the model to capture informative patterns in both categorical and numerical features while avoiding overfitting caused by small sample sizes.

3.2.3 Feature Tokenizer

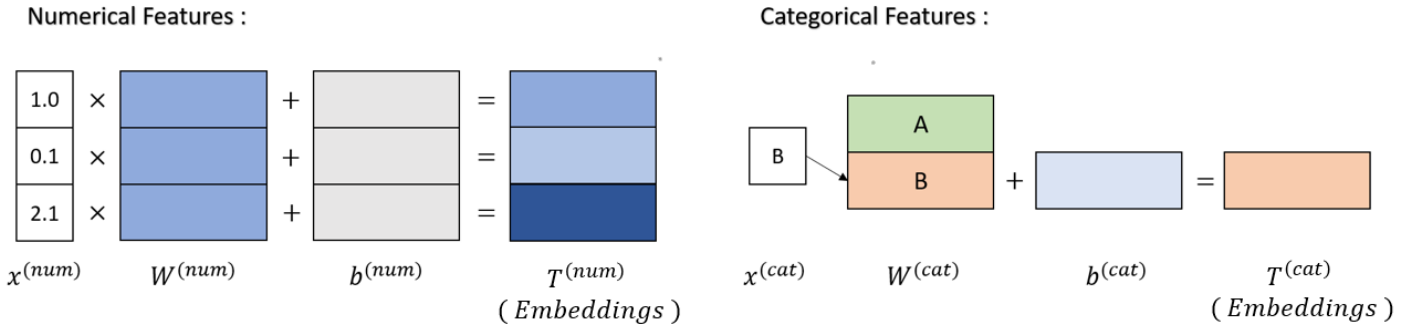


Figure 2: Feature Tokenizer

As shown in Figure 2, the *Feature Tokenizer* module transforms the input feature vector x into a sequence of learnable embeddings $T \in \mathbb{R}^{k \times d}$, where k denotes the total number of input features and d is the embedding dimension. Each feature x_j is individually projected into a d -dimensional vector using a learnable weight W and a bias term b .

Numerical Features

Each numerical feature $x_j^{(num)} \in \mathbb{R}$ is transformed into an embedding by multiplying by a learnable weight $W_j^{(num)} \in \mathbb{R}^d$, followed by an additive feature-specific bias term $b_j^{(num)} \in \mathbb{R}^d$.

The transformation is defined as:

$$T_j^{(\text{num})} = b_j^{(\text{num})} + x_j^{(\text{num})} \cdot W_j^{(\text{num})}$$

Here, $W_j^{(\text{num})}$ scales the scalar value x_j across all embedding dimensions, turning it into a high-dimensional vector. The bias term $b_j^{(\text{num})}$ adds a learnable shift, allowing the model to adjust the embedding regardless of the input value.

Categorical Features

Each categorical feature $x_j^{(\text{cat})}$, encoded as a one-hot vector $e_j \in \{0, 1\}^{S_j}$ where S_j is the number of unique categories, is mapped to a d -dimensional embedding via a lookup in a learnable embedding matrix $W_j^{(\text{cat})} \in \mathbb{R}^{S_j \times d}$, followed by the addition of a feature-specific bias term $b_j^{(\text{cat})} \in \mathbb{R}^d$. The transformation is:

$$T_j^{(\text{cat})} = b_j^{(\text{cat})} + e_j^\top W_j^{(\text{cat})}$$

In this case, $W_j^{(\text{cat})}$ represents a trainable embedding table, where each row corresponds to a unique category in feature x_j , and $b_j^{(\text{cat})}$ again provides a learnable offset.

In this framework, for each input sample, I generate two types of embeddings: one from the original raw features and another from the target-encoded features. Each of these is transformed into a sequence of feature tokens of shape $k \times d$, where k is the total number of features and d is the embedding dimension, and final output of the tokenizer is formed by stacking these two sets of embeddings along a new axis, resulting in a tensor of shape $2 \times k \times d$.

Formally, the original feature embedding is given by:

$$T = \text{stack} \left(T_1^{(\text{num})}, \dots, T_{k_{\text{num}}}^{(\text{num})}, T_1^{(\text{cat})}, \dots, T_{k_{\text{cat}}}^{(\text{cat})} \right) \in \mathbb{R}^{k \times d}$$

$$\tilde{T} = \text{concat} \left(T^{(\text{raw})}, T^{(\text{enc})} \right) \in \mathbb{R}^{2 \times k \times d}$$

where k_{num} and k_{cat} denote the number of numerical and categorical features, respectively.

3.3 Model Architecture

XDEAT block, a Transformer-based architecture, which integrates two key components: Cross-Dimensional Self-Attention and Cross-Encoding Self-Attention. The model takes the feature embeddings \tilde{T} as input, including both raw and target-encoded representations.

To enhance global context modeling, learnable, randomly initialized global tokens are appended to the input embeddings. These representations are then processed through multiple layers of Cross-Dimensional and Cross-Encoding Self-Attention modules.

Finally, the model generates the prediction based on the global tokens from the final layer.

Prerequisite: Self-Attention

Self-attention follows the scaled dot-product attention formulation. Given a query matrix \mathbf{Q} , key matrix \mathbf{K} , and value matrix \mathbf{V} , attention is computed as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax} \left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{h}} \right) \mathbf{V}$$

In self-attention, all three matrices are derived from the same input \mathbf{X} . The projection matrices $\mathbf{W}^Q, \mathbf{W}^K, \mathbf{W}^V \in \mathbb{R}^{h \times h}$ are learnable parameters.

Global Token Extension

To enable effective aggregation of information across all features and encoding views, a strategy inspired by the use of special tokens in Transformer models is adopted, such as the [CLS] token in BERT for sequence-level classification tasks. Specifically, learnable global tokens that act as aggregators within the attention mechanism are introduced.

For each input sample, a set of randomly initialized and trainable global tokens is appended to the input embedding tensor $\tilde{T} \in \mathbb{R}^{2 \times (k+1) \times d}$, where 2 denotes the two encoding views (raw and target-encoded), k is the number of features, 1 indicates the added global token per view, and d is the embedding dimension.

Cross-Dimensional Self-Attention

To model dependencies across feature dimensions within each encoding view, self-attention is applied separately to the raw and target-encoded representations. Following the global token extension described earlier, a learnable global token is inserted into the token sequence of each view to facilitate global context aggregation.

The token sequences for the raw and target-encoded views are referred to $\tilde{T}_{\text{raw}} \in \mathbb{R}^{(k+1) \times d}$ and $\tilde{T}_{\text{enc}} \in \mathbb{R}^{(k+1) \times d}$, respectively. Self-attention is then applied independently to each view:

$$\hat{T}_{\text{raw}} = \text{Self-Attention}(\tilde{T}_{\text{raw}}), \quad \hat{T}_{\text{enc}} = \text{Self-Attention}(\tilde{T}_{\text{enc}})$$

This mechanism enables each feature token to attend to all other tokens, including the global token, within its respective view, thereby capturing rich intra-view interactions and enhancing

global information flow.

Cross-Encoding Self-Attention

While the previous step models dependencies across feature dimensions, this stage captures interactions between the two encoding views (raw and target-encoded).

To achieve this, the two views are treated as parallel sequences of tokens, where each pair of tokens at the same feature position across the two views are semantically aligned. Given the processed representations from the previous self-attention step, denoted as $\hat{T}_{\text{raw}} \in \mathbb{R}^{(k+1) \times d}$ and $\hat{T}_{\text{enc}} \in \mathbb{R}^{(k+1) \times d}$, these representations are stacked along a new dimension to form a tensor $\hat{T} \in \mathbb{R}^{2 \times (k+1) \times d}$.

Then, self-attention is applied across the encoding dimension (between the raw and target-encoded views) for each feature position. This allows each feature to integrate complementary signals from both views. Formally, for each feature index $i \in \{1, \dots, k+1\}$:

$$\hat{T}^{(i)} = \text{SelfAttn} \left(\begin{bmatrix} \hat{T}_{\text{raw}}^{(i)} \\ \hat{T}_{\text{enc}}^{(i)} \end{bmatrix} \right) \in \mathbb{R}^{2 \times d}$$

where $\hat{T}_{\text{raw}}^{(i)}$ and $\hat{T}_{\text{enc}}^{(i)}$ are the tokens for the feature i in the raw and encoded views, respectively.

This attention mechanism enables the model to learn richer representations that combine both raw semantics and supervised statistical patterns.

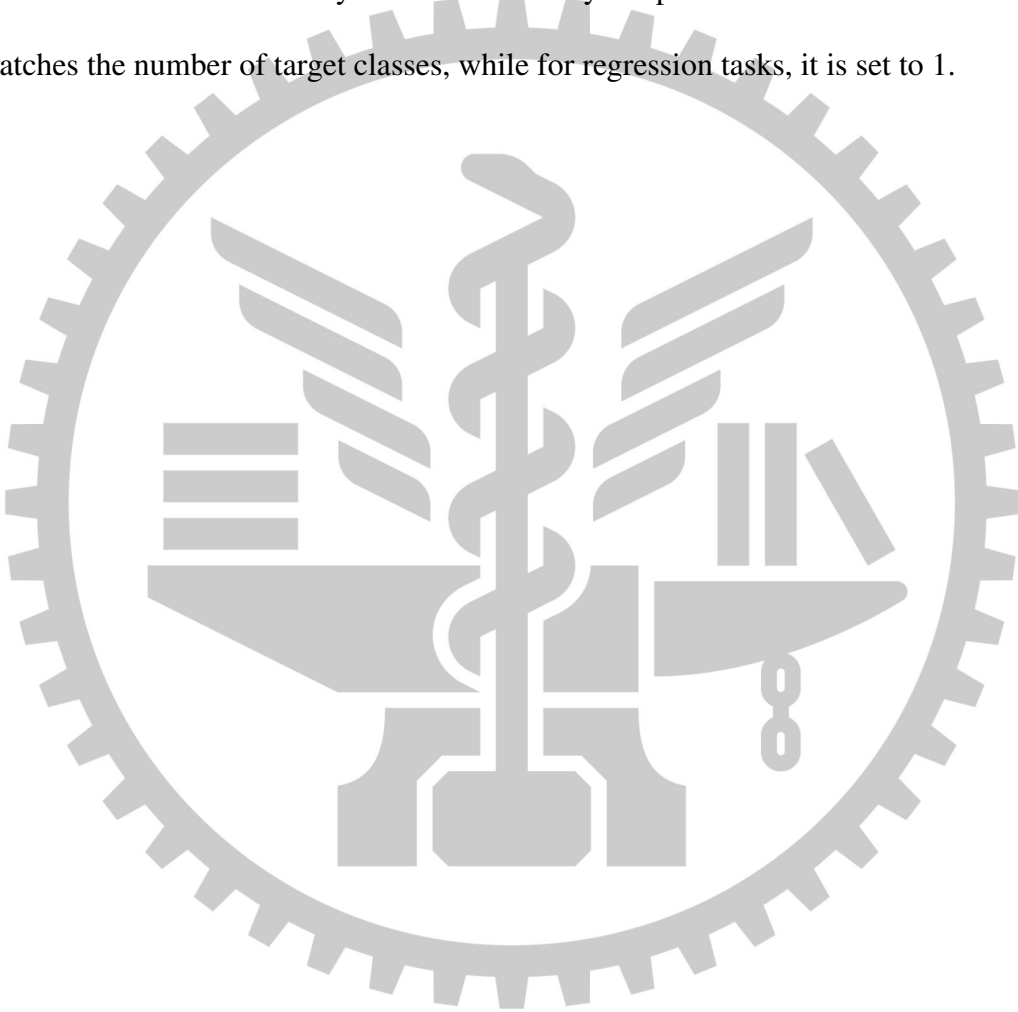
Final Prediction

After passing through multiple layers of Cross-Dimensional and Cross-Encoding Self-Attention, the global tokens act as comprehensive aggregators that capture both intra-view feature interactions and cross-view dependencies between the raw and target-encoded representations.

The global tokens are extracted from the final output of the layers, one from the raw view and one from the target-encoded view, and concatenated to form \mathbf{z} :

$$\mathbf{z} = [\hat{g}_{\text{raw}}; \hat{g}_{\text{enc}}] \in \mathbb{R}^{2d}$$

This vector \mathbf{z} is then passed through an MLP to produce the final output logits, and the output dimension of the final MLP layer is determined by the prediction task: for classification tasks, it matches the number of target classes, while for regression tasks, it is set to 1.



Chapter 4. Experiment Settings

4.1 Datasets

To comprehensively assess our model, we select a diverse set of benchmark datasets guided by prior work in tabular learning benchmarks [21]. These datasets span three major prediction tasks, regression, binary classification, and multiclass classification, and vary in feature composition: some contain only numerical features (CA, HI, JA), others contain only categorical (KDD), and some contain a mix of both (AD). This design ensures a balanced evaluation of the model’s ability to generalize across different real-world scenarios. Since our experiments are primarily conducted on an RTX 4090 GPU with 24 GB of memory, we focus on datasets of moderate size to remain within memory limits while still covering diverse tasks. To further evaluate scalability, we also include the Jannis dataset [31], which contains substantially more samples and features; this dataset is trained using an H100 GPU to verify that our model can effectively handle large-scale tabular learning. For detailed dataset statistics, please refer to Table 1.

Table 1: Datasets description

Name	Abbr	OpenML ID	# Train	# Validation	# Test	# Num	# Cat	Task type	Label Distribution
California Housing	CA	-	13209	3303	4128	8	0	Regression	—
Adult	AD	1590	26048	6513	16281	6	8	Binary classification	0: 76%, 1: 24%
KDD Internet Usage	KDD	981	5390	1348	3370	0	68	Binary classification	0: 73%, 1: 27%
Higgs Small	HI	23512	62752	15688	19610	28	0	Binary classification	1: 53%, 0: 47%
Jannis	JA	41168	53588	13398	16747	54	0	Multiclass classification	0: 2%, 1: 34%, 2: 18%, 3: 46%

4.1.1 California Housing

The California Housing dataset [32] pertains to houses found in a given California district and some summary statistics on them based on 1990 census data. The final data contained 20,640 observations on 8 numerical variables. The dependent variable is median house value.

4.1.2 Adult

The Adult dataset [33] is a benchmark dataset commonly used for classification tasks in machine learning. It contains 48,842 instances with 14 features, which include both categorical and integer types. The goal is to predict whether an individual's annual income exceeds 50,000, based on census information such as age, education level, occupation, and marital status.

4.1.3 KDD Internet Usage

The KDD Internet Usage dataset [34], originally compiled by the Georgia Tech Research Corporation as part of the GVVU's WWW User Surveys, provides detailed information on users' internet usage patterns and demographic characteristics. Notably, the dataset consists exclusively of categorical features. The objective of this task is to predict which users are likely to pay for internet access at work based on these features.

4.1.4 HIGGS Small

The HIGGS dataset [35] is a high-energy physics dataset generated via Monte Carlo simulations. It contains around 98K samples with 21 low-level features and 7 high-level features. All features are numerical types and are used for classifying particle collision events.

4.1.5 Jannis

The Jannis dataset [31] is a tabular classification benchmark from the OpenML repository, commonly used in evaluating deep learning models for structured data. It contains 54 numerical features, and the objective is to predict a multi-class target variable.

4.2 Environment Setup

All experiments were conducted using Python 3.10 and PyTorch 2.5.1. Training was primarily performed on an NVIDIA RTX 4090 GPU with 24GB of memory. However, due to the high memory demand of Optuna’s hyperparameter search, the RTX 4090 was insufficient for running the Jannis dataset. As a result, experiments on the Jannis dataset were additionally executed on an NVIDIA H100 GPU. To maintain reproducibility, random seeds were fixed across NumPy, PyTorch, and other relevant libraries.

4.2.1 Optuna

Optuna library [36] is a Python-based framework that enables automated exploration of hyperparameter spaces. In this study, multi-objective optimization is used to maximize both training and validation performance. This dual-objective formulation was designed to strike a balance between model fitting capacity (i.e., high training performance) and generalization ability (i.e., high validation performance). The evaluation metrics used for optimization varied by dataset and task; The entire list of hyperparameters considered during optimization, along with their search ranges, is provided in **Appendix ((Optuna) Model Hyper-Parameter Configuration)**.

4.2.2 Implementation Details

All models were trained using the AdamW optimizer. A cosine annealing learning rate scheduler was applied, with a linear warm-up phase during the first 5 epochs. Training was subject to a maximum of 200 epochs, with early stopping applied if validation performance did not improve for 15 consecutive epochs. The actual number of training epochs varied by dataset and is summarized in **Appendix (Table 16)**, reflecting the dynamic behavior of the early stopping mechanism within the 200-epoch limit. For hyperparameter tuning, the random seed was fixed at 42 to ensure reproducibility. Once the best configuration was identified, each experiment was independently repeated 15 times with different random seeds, and the average performance was reported to account for variability.

4.2.3 Evaluation Metrics

For classification tasks, we primarily follow the evaluation approach in [21], using Accuracy and RMSE as performance metrics. However, given that the AD, KDD, and JA datasets exhibit class imbalance, we also employ additional metrics, such as AUROC, AUPRC, F1 Score, Recall, and Precision, to provide a more comprehensive evaluation of the model’s performance. Detailed metric performance can be found in the **Appendix (Table 15)**.

4.2.4 Preprocessing

Before passing the input into the feature tokenizer, different preprocessing strategies are applied to the raw and target-encoded features. For the raw values, the PowerTransformer with the Yeo-Johnson method is used to stabilize variance and make the data more Gaussian-like. For the target-encoded features, StandardScaler is applied to center and scale the values.

For regression problems, the target values are standardized:

$$y_{\text{new}} = \frac{y_{\text{old}} - \text{mean}(y_{\text{train}})}{\text{std}(y_{\text{train}})} \quad (4.1)$$

4.2.5 Benchmark Model

The benchmark models were selected based on their inclusion in previous studies [21, 37], in order to ensure consistency and comparability with prior works. A brief description of each model is provided below:

- **XGBoost**: A classic and robust gradient boosting tree method [15].
- **MLP**: A simple multilayer perceptron, used as a basic benchmark for tabular tasks [21].
- **ResNet**: A deep neural network architecture with residual connections, known for its stable training in deep architectures [21].
- **DCN-V2**: A model that combines explicit and implicit feature interaction mechanisms, particularly designed for recommendation tasks [27].
- **NODE**: A tree-based neural model with an ensemble-like structure, showing competitive performance on various tabular benchmarks [28].
- **AutoInt**: A neural network model that employs multi-head self-attention mechanisms to automatically learn high-order feature interactions in tabular data [20].
- **FT-Transformer**: A Pre-Norm transformer model that uses a feature tokenizer [21].
- **XDEAT**: Proposed method, which integrates dynamic encoding strategies and two-stage attention to enhance performance.

Chapter 5. Results

5.1 Main Results

In this study, all baseline models are trained and validated under a unified experimental setting based on Section 4.2 to ensure fairness and reproducibility. In addition, hyperparameter optimization for all models is conducted using the Optuna framework. The detailed search spaces and parameter settings are provided in the **Appendix (Optuna) Model Hyper-Parameter Configuration** for reproducibility. Furthermore, a one-sided Wilcoxon Test is conducted to statistically assess the significance of the differences between XDEAT and the other models, with the detailed results provided in the **Appendix (Table 13)**.

Based on the results shown in Table 13, the proposed model, XDEAT, achieves the best overall performance with the lowest average rank of **1.80** and a standard deviation of **0.45**, clearly demonstrating its superiority and stability over other benchmark models.

Furthermore, the results are consistent with findings reported in the FT-Transformer paper, where XGBoost demonstrates strong performance on certain datasets. Similarly, ResNet and FT-Transformer exhibit competitive results [21]. Among all deep learning methods, XDEAT clearly stands out by achieving near state-of-the-art performance across almost all datasets.

The relatively low standard deviation in XDEAT’s rank (0.45) further emphasizes its stability and robustness across different datasets. These results demonstrate that our model not only excels in different tasks but also exhibits high reliability and generalization capability, making it a strong candidate for tabular data tasks.

Table 2: Comparison of performance across various benchmark models. **Bold** indicates the best performance across all models; underline indicates the second-best performance across all models.

Datasets	CA	AD	HI	KDD	JA	–
Metrics	RMSE↓	Accuracy↑				Rank
XGBoost	0.462±0.013	0.859±0.002	0.709±0.003	<u>0.900±0.004</u>	0.701±0.002	4.00±3.67
MLP	0.599±0.009	0.850±0.002	0.724±0.003	0.892±0.002	0.715±0.004	6.00±0.71
Resnet	0.588±0.009	0.851±0.002	0.734±0.003	0.892±0.006	0.726±0.004	3.20±1.79
DCN-V2	0.608±0.033	0.850±0.002	0.725±0.003	0.899±0.007	0.714±0.004	5.80±1.48
NODE	0.592±0.009	0.851±0.002	0.724±0.003	0.892±0.005	0.711±0.004	5.40±1.52
AutoInt	0.593±0.011	0.850±0.002	0.730±0.003	<u>0.900±0.006</u>	0.717±0.002	4.20±1.79
FT-Transformer	0.589±0.010	0.851±0.002	0.730±0.002	0.899±0.005	0.728±0.003	<u>3.00±1.22</u>
XDEAT	<u>0.508±0.011</u>	<u>0.857±0.002</u>	<u>0.733±0.003</u>	0.902±0.004	<u>0.727±0.002</u>	1.80±0.45

5.2 Ablation Studies

Target Encoding in Deep Learning

To examine the effect of different target encoding strategies, I evaluate the FT-Transformer under various encoding schemes, as shown in Table 3. The basic configuration uses raw values, meaning no encoding is applied, while other variants employ target encoding techniques such as Mean encoding, S-Shrink, M-Estimate, and Beta encoding. These methods rely on different smoothing techniques, with detailed implementation provided in the **Appendix (Details of**

⁰Values in Table 2, 3, 4, 5, 6 are reported in the format of **mean ± standard deviation**.

Target Encoders).

Table 3: Performance of Various Target Encoding Methods with FT-Transformer. **Bold** indicates the best performance across all models; underline indicates the second-best performance across all models.

Encoding Method	Dataset	CA	AD	HI	KDD	JA	–
	Metrics	RMSE ↓			Accuracy ↑		Rank
Mean Target Encoding		0.571 ± 0.014	0.857 ± 0.003	0.701 ± 0.004	0.901 ± 0.005	0.717 ± 0.003	3.00 ± 1.22
S-Shrink Target Encoding		0.567 ± 0.015	0.858 ± 0.002	0.700 ± 0.003	0.901 ± 0.005	0.718 ± 0.003	<u>1.80 ± 0.84</u>
Beta Target Encoding		0.568 ± 0.018	0.858 ± 0.003	0.697 ± 0.004	0.902 ± 0.004	0.716 ± 0.004	2.60 ± 1.34
M-Estimate Target Encoding		0.568 ± 0.018	0.859 ± 0.002	0.699 ± 0.003	0.902 ± 0.005	0.718 ± 0.003	1.60 ± 0.89
No Encoding (Raw Value)		0.593 ± 0.013	0.850 ± 0.002	0.730 ± 0.004	0.904 ± 0.006	0.725 ± 0.003	–
M-Estimate + Raw Value		0.496 ± 0.008	0.859 ± 0.003	0.727 ± 0.003	0.902 ± 0.006	0.719 ± 0.004	–

The results indicate that no single target encoding technique consistently outperforms others across all datasets, and the differences between the techniques are relatively minor. Based on the rankings, M-Estimate Target Encoding is selected in our approach, as it achieved the highest rank among the evaluated techniques.

Furthermore, a variation in performance between raw values and target encoding was observed. For example, on the HIGGS dataset, the raw values achieve a score of 0.730, while target encoding only reaches 0.701. In contrast, on the Adult dataset, target encoding performs better with a score of 0.859, compared to 0.850 for raw values. These results suggest that the information provided by target encoding and raw values differs.

Moreover, the M-Estimate + Raw Value setting achieves the best RMSE on the CA dataset but fails to consistently outperform either M-Estimate Target Encoding or Raw Value alone on the other benchmarks. This suggests that simple concatenation of raw and encoded features does not guarantee better performance and may be insufficient for capturing complex depen-

dencies. These results highlight the need for more advanced architectural designs—such as the proposed XDEAT—which explicitly model inter- and intra-feature relationships to fully utilize the complementary strengths of raw and encoded inputs.

Target Encoding in Machine Learning

To verify that the improvements introduced by target encoding are not confined to a specific model architecture, I further extend our analysis by applying the same encoding strategies to a non-transformer baseline, XGBoost, as shown in Table 4. This allows us to assess the generalizability and robustness of various target encoding methods across different model families. The XGBoost model is trained using fixed hyperparameters (`booster="gbtree"`, `early-stopping-rounds=50`, and `n-estimators=2000`).

Table 4: Performance of Various Target Encoding Methods with XGBoost. **Bold** indicates the best performance across all models; underline indicates the second-best performance across all models.

Encoding Method	Dataset	CA	AD	HI	KDD	JA	–
		RMSE ↓			Accuracy ↑		Rank
Mean Target Encoding		0.474 ± 0.025	0.863 ± 0.003	0.695 ± 0.003	0.894 ± 0.004	0.704 ± 0.003	2.20 ± 1.30
S-Shrink Target Encoding		0.459 ± 0.011	0.863 ± 0.003	0.695 ± 0.004	0.895 ± 0.005	0.705 ± 0.003	1.00 ± 0.00
Beta Target Encoding		0.460 ± 0.010	0.863 ± 0.002	0.694 ± 0.003	0.894 ± 0.004	0.703 ± 0.004	2.60 ± 1.14
M-Estimate Target Encoding		0.462 ± 0.010	0.863 ± 0.002	0.695 ± 0.004	0.895 ± 0.006	0.703 ± 0.003	<u>1.80 ± 1.10</u>
No Encoding (Raw Value)		0.417 ± 0.011	0.864 ± 0.003	0.718 ± 0.004	0.897 ± 0.006	0.713 ± 0.003	–
M-Estimate + Raw Value		0.414 ± 0.007	0.870 ± 0.002	0.715 ± 0.002	0.895 ± 0.005	0.706 ± 0.003	-

Based on the results presented in Table 4, the performance of different target encoding techniques is relatively similar. Under the XGBoost setting, S-Shrink achieves the highest overall rank, followed closely by M-Estimate. Since the architecture of XDEAT is more closely

similar with FT-Transformer, the choice of M-Estimate encoding is guided by the results in Table 3, rather than those from Table 4, which reflect the performance under XGBoost.

Interestingly, none of the target encoding methods consistently outperforms the raw input across all datasets. While the AD and KDD datasets show comparable performance between encoded and non-encoded inputs, there is a noticeable performance gap in the CA and HI datasets, where the No Encoding baseline performs better. This highlights that the effectiveness of target encoding can be dataset-dependent and may not lead to improvements under XGBoost.

Comparison of Bin-Based and Non-Binned Strategies for Numerical Features

To evaluate the impact of bin-based preprocessing on numerical feature encoding, we compare two variants of the M-Estimate Encoding method: a non-binned version and a bin-based alternative. The evaluation is conducted on three tabular datasets—CA, HI, and JA—specifically chosen because they contain only numerical features, enabling a focused analysis without the confounding effects of categorical variables. As shown in Table 5, both encoding strategies are assessed in terms of model performance and encoding time (excluding training). The non-binned approach computes smoothed target statistics directly from raw feature values, while the bin-based method introduces discretization prior to encoding in order to reduce computational overhead and enhance robustness, particularly for large-scale datasets. The results reveal a trade-off between predictive performance and preprocessing efficiency.

Table 5: Comparison of performance and encoding time (averaged over 15 experiments, excluding model training) between non-binned and bin-based encoding strategies under M-Estimate Encoding for numerical features across three datasets: CA, HI, and JA.

Binning Strategy	Dataset	CA		HI		JA	
	Metrics	RMSE↓	Time (s)	Accuracy↑	Time (s)	Accuracy↑	Time (s)
No Binning		0.512 ± 0.009	7.460	0.733 ± 0.003	672.779	0.726 ± 0.003	935.085
Bin-Based		0.508 ± 0.011	0.412	0.733 ± 0.003	9.968	0.727 ± 0.002	21.996

Based on the results in Table 5, the bin-based strategy achieves comparable performance while drastically reducing encoding time. This highlights the effectiveness of the binning technique, making it an essential step for scaling M-Estimate encoding on numerical features.

Performance of self-attention components

The main components of XDEAT are Cross-Dimensional Self-Attention and Cross-Encoding Self-Attention, each contributing uniquely to the performance of the model. To evaluate the effectiveness of these components, a comparative analysis is conducted by testing the model with different configurations.

Additionally, the performance of these two components against a simpler architecture is evaluated using simple concatenation, which combines the feature representations directly. This comparison helps to assess whether the model architecture provides a significant advantage over basic concatenation with self-attention.

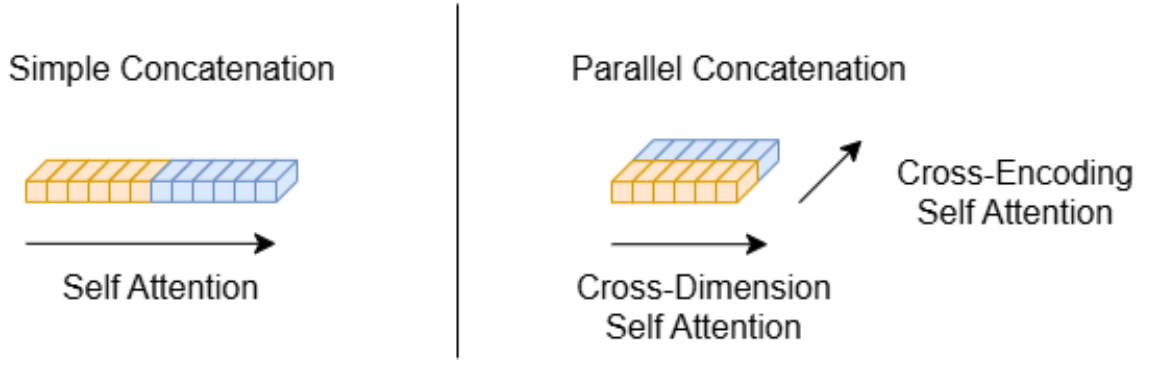


Figure 3: Ablation.

Table 6: Performance comparison under different model components. **Bold** indicates the best performance across all models; underline indicates second-best performance across all models.

Model Component	Dataset	CA	AD	HI	KDD	JA
		RMSE ↓		Accuracy ↑		
Cross-Dim		<u>0.527 ± 0.012</u>	0.857 ± 0.003	<u>0.731 ± 0.003</u>	0.890 ± 0.004	0.724 ± 0.003
Cross-Dim + Cross-Enc		0.508 ± 0.011	0.857 ± 0.003	0.733 ± 0.003	0.904 ± 0.006	0.727 ± 0.002
Self-Attention (Simple Concatenation)		0.531 ± 0.010	0.856 ± 0.003	<u>0.731 ± 0.003</u>	<u>0.900 ± 0.006</u>	<u>0.725 ± 0.002</u>

As shown in Table 14, incorporating both Cross-Dimensional and Cross-Encoding Self-Attention modules yields the best overall performance across most datasets. Notably, this configuration outperforms both the Cross-Dimensional-only variant and the simple concatenation baseline. These results highlight the complementary benefits of modeling intra-feature dependencies and inter-view interactions in improving predictive performance. To statistically validate the significance of the performance differences, a one-sided Wilcoxon Test is conducted to compare the best-performing model (Cross-Dimensional + Cross-Encoding) against the other configurations. The detailed test results are provided in the **Appendix (Table 14)**.

Chapter 6. Discussion and Limitations

6.1 Discussion

As shown in Table 3, incorporating target encoding improves performance on specific datasets, particularly AD and CA, where gradient boosting methods like XGBoost also demonstrate strong performance. Moreover, while different variants of target encoding (e.g., mean, smoothed, M-estimate) yield relatively comparable results, the overall benefit appears dataset-dependent. Furthermore, the target-encoded binning strategy for numerical values strikes a balance between preserving predictive information and reducing computation cost (Table 5).

Interestingly, as presented in Table 4, applying target-encoded inputs directly to XGBoost leads to a degradation in performance. This may be attributed to the incompatibility between the continuous values produced by target encoding and XGBoost’s tree-based structure.

More importantly, the ablation study in Table 14 highlights the effectiveness of our proposed model architecture. The Cross-Dim ONLY module, which captures intra-feature dependencies across embedding dimensions, achieves competitive performance compared to simple concatenation on several datasets, including CA, AD, and HI. Moreover, the full Cross-Dim with Cross-Enc architecture, which jointly models both inter-feature and inter-dimensional interactions through attention mechanisms, yields superior performance on datasets such as CA, KDD and JA. These results further support the advantage of explicitly modeling feature dependencies and an encoding perspective to enhance representation learning.

6.2 Limitations

One limitation of this thesis lies in the use of the traditional self-attention mechanism. While effective, traditional self-attention is known to be computationally expensive, particularly for long sequences [22]. In recent years, several alternatives have been proposed to improve memory efficiency and inference speed, such as Linear Transformer [38] and Transformer-XL [39]. For instance, the implementation of FT-Transformer [21] adopts Linear Transformer [38] as its backbone to significantly reduce computational overhead [22]. Incorporating such techniques into our model could potentially improve both efficiency and scalability.

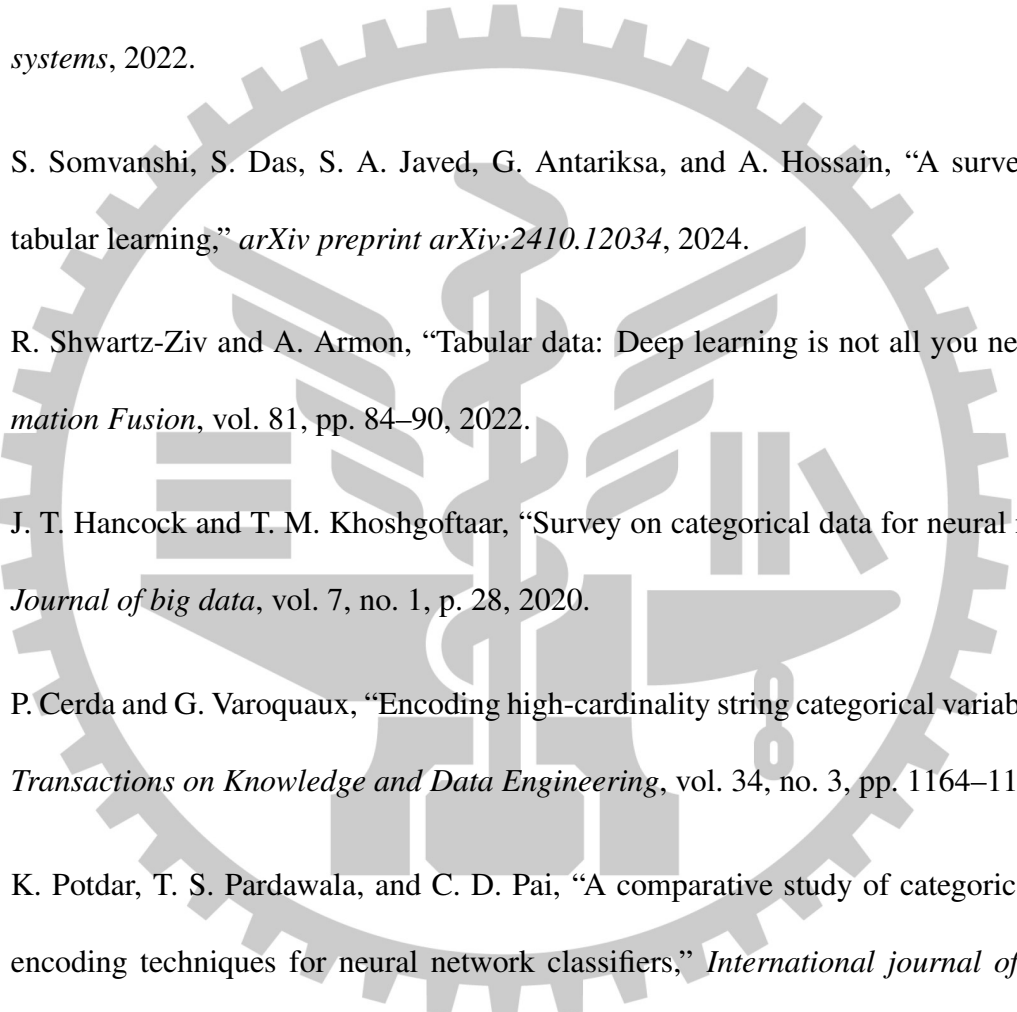
Another limitation of our model is its sensitivity to irrelevant or noisy features. This issue is particularly pronounced in deep learning models for tabular data, which are often susceptible to performance degradation when exposed to uninformative inputs [13]. To mitigate this problem and improve computational efficiency, several methods have been proposed. For instance, CrossFormer [29] acknowledges this challenge in its conclusion and suggests that future research could incorporate GNN-based structure learning, such as the approach proposed in NodeFormer [40]. Moreover, AMFormer [41] addresses feature selection by employing prompt tokens and a top-k selection strategy to dynamically identify important feature interactions. These efforts highlight the ongoing need to develop mechanisms that can effectively focus on informative features while minimizing the influence of irrelevant ones.

Chapter 7. Conclusions

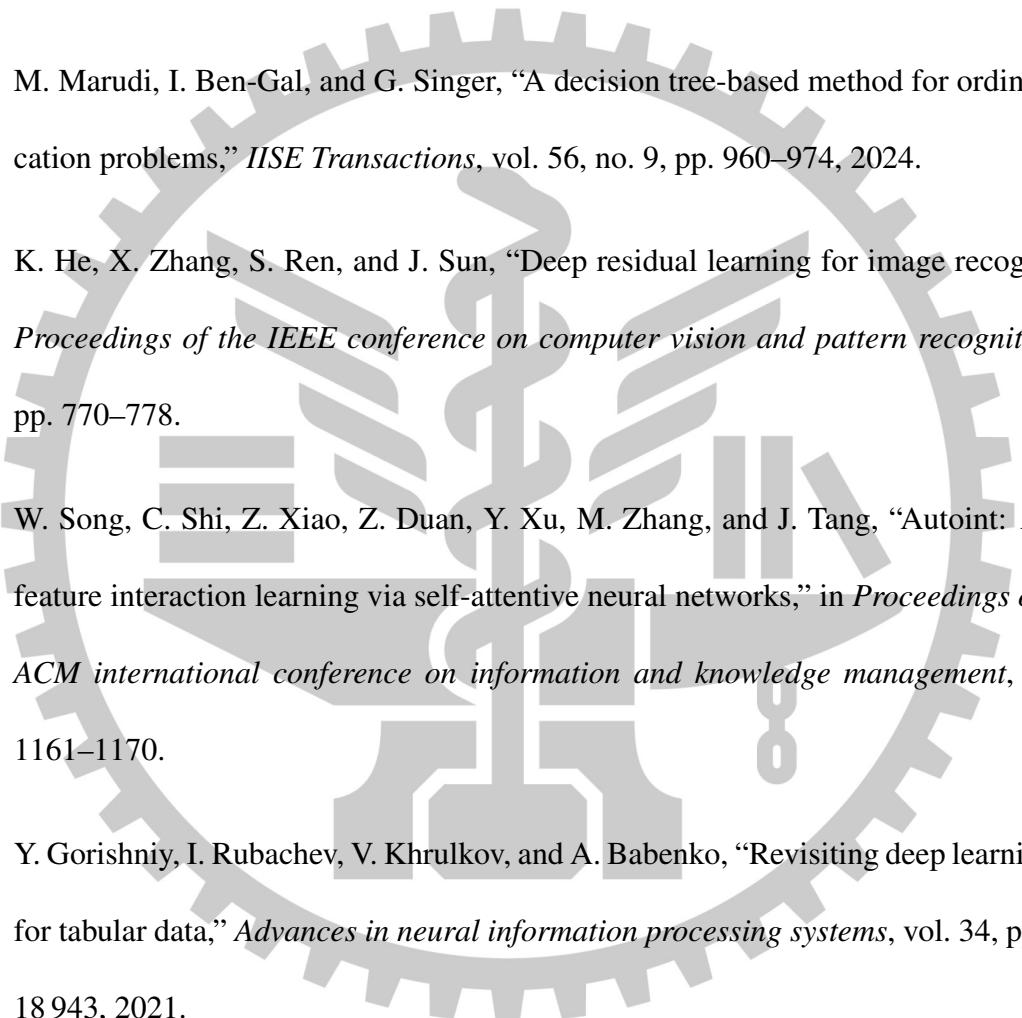
This work presents a novel framework that enhances tabular data modeling through two major contributions: a bin-based target encoding method for numerical features and a dual-view attention architecture that captures both intra-view and inter-view interactions. The proposed binning-based encoder effectively captures the target distribution of numerically similar values with minimal computational overhead, while the architectural design leverages Cross-Dimension and Cross-Encoding Self-Attention to model complex feature relationships.

Experiments across multiple tabular benchmarks demonstrate that the proposed method consistently outperforms strong baselines. As shown in Table 13, the proposed model, XDEAT, achieves the best average rank. These results highlight the effectiveness of both the encoding strategy and the architectural design in improving performance on tabular data.

References

- 
- [1] V. Borisov, T. Leemann, K. Seßler, J. Haug, M. Pawelczyk, and G. Kasneci, “Deep neural networks and tabular data: A survey,” *IEEE transactions on neural networks and learning systems*, 2022.
- [2] S. Somvanshi, S. Das, S. A. Javed, G. Antariksa, and A. Hossain, “A survey on deep tabular learning,” *arXiv preprint arXiv:2410.12034*, 2024.
- [3] R. Schwartz-Ziv and A. Armon, “Tabular data: Deep learning is not all you need,” *Information Fusion*, vol. 81, pp. 84–90, 2022.
- [4] J. T. Hancock and T. M. Khoshgoftaar, “Survey on categorical data for neural networks,” *Journal of big data*, vol. 7, no. 1, p. 28, 2020.
- [5] P. Cerda and G. Varoquaux, “Encoding high-cardinality string categorical variables,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 3, pp. 1164–1176, 2020.
- [6] K. Potdar, T. S. Pardawala, and C. D. Pai, “A comparative study of categorical variable encoding techniques for neural network classifiers,” *International journal of computer applications*, vol. 175, no. 4, pp. 7–9, 2017.
- [7] D. Micci-Barreca, “A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems,” *ACM SIGKDD explorations newsletter*, vol. 3, no. 1, pp. 27–32, 2001.

- [8] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, “Feature hashing for large scale multitask learning,” in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 1113–1120.
- [9] W. Yustanti and N. Iriawan, “Categorical encoder based performance comparison in pre-processing imbalanced multiclass classification,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 31, no. 3, pp. 1705–1715, 2023.
- [10] F. Pargent, F. Pfisterer, J. Thomas, and B. Bischl, “Regularized target encoding outperforms traditional methods in supervised machine learning with high cardinality features,” *Computational Statistics*, vol. 37, no. 5, pp. 2671–2692, 2022.
- [11] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.
- [12] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.
- [13] D. McElfresh, S. Khandagale, J. Valverde, V. Prasad C, G. Ramakrishnan, M. Goldblum, and C. White, “When do neural nets outperform boosted trees on tabular data?” *Advances in Neural Information Processing Systems*, vol. 36, pp. 76 336–76 369, 2023.
- [14] L. Grinsztajn, E. Oyallon, and G. Varoquaux, “Why do tree-based models still outperform deep learning on typical tabular data?” *Advances in neural information processing systems*, vol. 35, pp. 507–520, 2022.
- [15] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

- 
- [16] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, “Lightgbm: A highly efficient gradient boosting decision tree,” *Advances in neural information processing systems*, vol. 30, 2017.
- [17] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, “Catboost: unbiased boosting with categorical features,” *Advances in neural information processing systems*, vol. 31, 2018.
- [18] M. Marudi, I. Ben-Gal, and G. Singer, “A decision tree-based method for ordinal classification problems,” *IJSE Transactions*, vol. 56, no. 9, pp. 960–974, 2024.
- [19] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [20] W. Song, C. Shi, Z. Xiao, Z. Duan, Y. Xu, M. Zhang, and J. Tang, “AutoInt: Automatic feature interaction learning via self-attentive neural networks,” in *Proceedings of the 28th ACM international conference on information and knowledge management*, 2019, pp. 1161–1170.
- [21] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, “Revisiting deep learning models for tabular data,” *Advances in neural information processing systems*, vol. 34, pp. 18 932–18 943, 2021.
- [22] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, “Efficient transformers: A survey,” *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–28, 2022.

- [23] C.-T. Li, Y.-C. Tsai, C.-Y. Chen, and J. C. Liao, “Graph neural networks for tabular data learning: A survey with taxonomy and directions,” *arXiv preprint arXiv:2401.02143*, 2024.
- [24] M. Joseph and H. Raj, “Gandalf: gated adaptive network for deep automated learning of features,” *arXiv preprint arXiv:2207.08548*, 2022.
- [25] G. Liu, J. Yang, and L. Wu, “Ptab: Using the pre-trained language model for modeling tabular data,” *arXiv preprint arXiv:2209.08060*, 2022.
- [26] S. Hegselmann, A. Buendia, H. Lang, M. Agrawal, X. Jiang, and D. Sontag, “Tabllm: Few-shot classification of tabular data with large language models,” in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2023, pp. 5549–5581.
- [27] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi, “Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems,” in *Proceedings of the web conference 2021*, 2021, pp. 1785–1797.
- [28] S. Popov, S. Morozov, and A. Babenko, “Neural oblivious decision ensembles for deep learning on tabular data,” *arXiv preprint arXiv:1909.06312*, 2019.
- [29] Y. Zhang and J. Yan, “Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting,” in *The eleventh international conference on learning representations*, 2023.
- [30] W. Zhu, R. Qiu, and Y. Fu, “Comparative study on the performance of categorical variable encoders in classification and regression tasks,” *arXiv preprint arXiv:2401.09682*, 2024.

- [31] I. Guyon, L. Sun-Hosoya, M. Boullé, H. J. Escalante, S. Escalera, Z. Liu, D. Jajetic, B. Ray, M. Saeed, M. Sebag *et al.*, “Analysis of the automl challenge series,” *Automated Machine Learning*, vol. 177, pp. 177–219, 2019.
- [32] R. K. Pace and R. Barry, “Sparse spatial autoregressions,” *Statistics & Probability Letters*, vol. 33, no. 3, pp. 291–297, 1997.
- [33] R. Kohavi *et al.*, “Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid,” in *Kdd*, vol. 96, 1996, pp. 202–207.
- [34] C. M. Kehoe and J. E. Pitkow, “Surveying the territory: Gvu ’ s five www user surveys,” *The World Wide Web Journal*, vol. 1, no. 3, pp. 77–84, 1996.
- [35] P. Baldi, P. Sadowski, and D. Whiteson, “Searching for exotic particles in high-energy physics with deep learning,” *Nature communications*, vol. 5, no. 1, p. 4308, 2014.
- [36] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 2623–2631.
- [37] J. Wu, S. Chen, Q. Zhao, R. Sergazinov, C. Li, S. Liu, C. Zhao, T. Xie, H. Guo, C. Ji *et al.*, “Switchtab: Switched autoencoders are effective tabular learners,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 14, 2024, pp. 15 924–15 933.
- [38] S. Wang, B. Z. Li, M. Khabsa, H. Fang, and H. Ma, “Linformer: Self-attention with linear complexity,” *arXiv preprint arXiv:2006.04768*, 2020.
- [39] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, “Transformer-xl: Attentive language models beyond a fixed-length context,” *arXiv preprint arXiv:1901.02860*, 2019.

- [40] Q. Wu, W. Zhao, Z. Li, D. P. Wipf, and J. Yan, “Nodeformer: A scalable graph structure learning transformer for node classification,” *Advances in Neural Information Processing Systems*, vol. 35, pp. 27 387–27 401, 2022.
- [41] Y. Cheng, R. Hu, H. Ying, X. Shi, J. Wu, and W. Lin, “Arithmetic feature interaction is necessary for deep tabular learning,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 10, 2024, pp. 11 516–11 524.
- [42] M. Larionov, “Sampling techniques in bayesian target encoding,” *arXiv preprint arXiv:2006.01317*, 2020.



Appendix

7.1 (Optuna) Model Hyper-Parameter Configuration

For each model, the hyper-parameter search space is defined based on common practices [21] and optimized using Optuna. The search is performed over 50 iterations, and the best configuration is selected according to both training and validation performance.

7.1.1 XGBoost

Implementation The following hyperparameters are fixed and not tuned:

- `booster = "gbtree"`
- `early-stopping-rounds = 50`
- `n-estimators = 2000`

Table 7: XGBoost hyperparameter space for datasets.

Parameter	Distribution
Max depth	UniformInt[3, 10]
Min child weight	LogUniform[1e-8, 1e5]
Subsample	Uniform[0.5, 1]
Learning rate	LogUniform[1e-5, 1]
Col sample by level	Uniform[0.5, 1]
Col sample by tree	Uniform[0.5, 1]
Gamma	LogUniform[1e-8, 1e2]
Lambda	LogUniform[1e-8, 1e2]
Alpha	LogUniform[1e-8, 1e2]
# Iterations	50

7.1.2 MLP

Implementation The implementation is based on the official package provided by [21].

Table 8: MLP hyperparameter space for datasets.

Parameter	Distribution
# Layers	UniformInt[1, 8]
Layer size	UniformInt[1, 512]
Dropout	{0, Uniform[0, 0.5]}
Learning rate	LogUniform[1e-5, 1e-2]
Weight decay	{0, LogUniform[1e-6, 1e-3]}
Category embedding size	UniformInt[64, 512] (for AD only)
# Iterations	50

7.1.3 ResNet

Implementation The implementation is based on the official package provided by [21].

Table 9: ResNet hyperparameter space for datasets.

Parameter	Distribution
# Layers	UniformInt[1, 8]
Layer size	UniformInt[64, 512]
Hidden factor	Uniform[1, 4]
Hidden dropout	Uniform[0, 0.5]
Residual dropout	{0, Uniform[0, 0.5]}
Learning rate	LogUniform[1e-5, 1e-2]
Weight decay	{0, LogUniform[1e-6, 1e-3]}
Category embedding size	(AD only) UniformInt[64, 512]
# Iterations	50

7.1.4 DCN V2

Architecture DCN V2 has two variants: “stacked” and “parallel.” Following [21], the “parallel” variant is used due to its slightly better performance on large datasets.

Implementation The implementation is based on the official codebase provided by [21].

Table 10: DCN V2 hyperparameter space for datasets.

Parameter	Distribution
# Cross layers	UniformInt[1, 8]
# Hidden layers	UniformInt[1, 8]
Layer size	UniformInt[64, 512]
Hidden dropout	Uniform[0, 0.5]
Cross dropout	Uniform[0, 0.5]
Learning rate	LogUniform[1e-5, 1e-2]
Weight decay	LogUniform[1e-6, 1e-3]
Category embedding size	UniformInt[64, 512]
# Iterations	50

7.1.5 NODE

Implementation The NODE model is configured with the following fixed settings due to hardware constraints:

- `layers = 1`
- `layer_dim = 2048`
- `depth = 6`
- `tree_dim = #classes` (set to 1 for regression)
- `category_embedding_size = 192`

7.1.6 AutoInt

Implementation The implementation is based on the official codebase provided by [21]. The number of attention heads is set to 2, as this configuration was found to be competitive.

Table 11: AutoInt hyperparameter space for datasets.

Parameter	Distribution
# Layers	UniformInt[1, 6]
Feature embedding size	UniformInt[8, 64]
Residual dropout	Uniform[0.0, 0.2]
Attention dropout	Uniform[0.0, 0.5]
Learning rate	LogUniform[1e-5, 1e-3]
Weight decay	LogUniform[1e-6, 1e-3]
# Iterations	50

7.1.7 FT-Transformer

Implementation The implementation is based on the official package provided by [21].

Table 12: FT-Transformer hyperparameter space for datasets.

Parameter	Distribution
# Layers	UniformInt[1, 4]
Feature embedding size	UniformInt[64, 512]
Residual dropout	Uniform[0, 0.2]
Attention dropout	Uniform[0, 0.5]
FFN dropout	Uniform[0, 0.5]
FFN factor	Uniform[2/3, 8/3]
Learning rate	LogUniform[1e-5, 1e-3]
Weight decay	LogUniform[1e-6, 1e-3]
# Iterations	50

7.2 Details of Target Encoders

This section presents the details of the target encoding methods used in this study. All encoders transform categorical (or discretized numerical) features into continuous values using statistics derived from the target variable. Let \bar{y}_c denote the global mean of the target, \bar{y}_j the mean target for category j , and n_j the number of samples in category j .

7.2.1 Mean Encoder

The Mean Encoder replaces each categorical value with the mean of the target variable within that category. This encoder is simple and effective, but can lead to overfitting, especially for categories with low sample counts. It is defined as:

$$\hat{y}_j = \bar{y}_j = \frac{1}{n_j} \sum_{i:x_i=j} y_i$$

7.2.2 S-Shrink Encoder

The S-Shrink encoder applies a smooth interpolation between the global mean \bar{y}_c and the category-specific mean \bar{y}_j , using a confidence weight B_j computed via a sigmoid function. The encoded value for category j is given by:

$$\hat{y}_j = (1 - B_j) \cdot \bar{y}_c + B_j \cdot \bar{y}_j \quad (7.1)$$

where the weight $B_j \in [0, 1]$ is defined as:

$$B_j = \frac{1}{1 + \exp\left(-\frac{n_j - S_1}{S_2}\right)} \quad (7.2)$$

Here, S_1 is a threshold indicating the number of instances required for reliable estimation, and S_2 controls the steepness of the transition. When $n_j \ll S_1$, the encoder relies more on the global mean \bar{y}_c ; when $n_j \gg S_1$, it favors the category mean \bar{y}_j . Following the configuration recommended in [30], the parameters were set as $S_1 = 20$ and $S_2 = 10$.

7.2.3 Beta Encoder

The Beta Encoder leverages Bayesian smoothing based on the Beta-Binomial model. For binary classification, it estimates the probability of the positive class using updated posterior parameters derived from both global and category-specific statistics.

The prior parameters are defined as:

$$\alpha = 1 + \gamma \sum_{n=1}^N y_n, \quad \beta = 1 + \gamma \sum_{n=1}^N (1 - y_n)$$

For a given category v , the posterior parameters are:

$$\alpha_v = \alpha + \sum_{x_n=v} y_n, \quad \beta_v = \beta + \sum_{x_n=v} (1 - y_n)$$

The final smoothed target estimate is computed as:

$$\hat{y}_v = \frac{\alpha_v}{\alpha_v + \beta_v}$$

Following [42], the parameter γ controls the strength of the prior, with higher values enforcing more smoothing.

7.2.4 M-Estimate Encoder

The M-Estimate Encoder smooths the category mean with the global mean using a fixed prior sample size m . The formula is:

$$\hat{y}_j = \frac{n_j \cdot \bar{y}_j + m \cdot \bar{y}_c}{n_j + m}$$

This approach offers a simple yet effective balance between the empirical category mean and the global prior, and is commonly used in practical applications.

7.3 Wilcoxon Test on Main Results

This section summarizes the performance of various benchmark models across five datasets. To assess the statistical significance of performance differences, the one-sided Wilcoxon test is applied with a significance level of $\alpha = 0.05$, adjusted with Bonferroni correction. This provides a measure of whether improvements are consistent and not due to random variation.

Table 13: Comparison of performance across various benchmark models. The best result and those not statistically different from it are all shown in **bold**. Values are reported as mean \pm standard deviation.

Datasets	CA	AD	HI	KDD	JA
Metrics	RMSE \downarrow	Accuracy \uparrow			
XGBoost	0.462\pm0.013	0.859\pm0.002	0.709 \pm 0.003	0.900\pm0.004	0.701 \pm 0.002
MLP	0.599 \pm 0.009	0.850 \pm 0.002	0.724 \pm 0.003	0.892 \pm 0.002	0.715 \pm 0.004
Resnet	0.588 \pm 0.009	0.851 \pm 0.002	0.734\pm0.003	0.892 \pm 0.006	0.726\pm0.004
DCN-V2	0.608 \pm 0.033	0.850 \pm 0.002	0.725 \pm 0.003	0.899\pm0.007	0.714 \pm 0.004
NODE	0.592 \pm 0.009	0.851 \pm 0.002	0.724 \pm 0.003	0.892 \pm 0.005	0.711 \pm 0.004
AutoInt	0.593 \pm 0.011	0.850 \pm 0.002	0.730 \pm 0.003	0.900\pm0.006	0.717 \pm 0.002
FT-Transformer	0.589 \pm 0.010	0.851 \pm 0.002	0.730 \pm 0.002	0.899\pm0.005	0.728\pm0.003
XDEAT	0.508 \pm 0.011	0.857\pm0.002	0.733\pm0.003	0.902\pm0.004	0.727\pm0.002

Table 13 highlights models that achieve the best or statistically comparable performance across five benchmark datasets, with bolded entries indicating results not significantly different from the best under the one-sided Wilcoxon test with Bonferroni correction. On the CA regression task, XGBoost achieves the lowest RMSE (0.462) and stands as the best performer. Although XDEAT does not outperform XGBoost, it achieves the best result among deep learning models (0.508), showing its strong regression capability. For the AD classification dataset, both XGBoost and XDEAT are statistically tied at the top, suggesting that XDEAT can match the performance of traditional gradient boosting methods on structured data. In the HI dataset,

XDEAT achieves top-tier performance alongside ResNet and FT-Transformer.. On the KDD dataset, an imbalanced classification task, XDEAT attains the highest accuracy (0.902) and is statistically equivalent to other strong baselines such as XGBoost, DCN-V2, AutoInt, and FT-Transformer. Lastly, for the JA multi-class classification task, FT-Transformer, Resnet, and XDEAT are the only models achieving statistically best performance, with nearly identical accuracies (0.728, 0.726, and 0.727, respectively).

7.4 Wilcoxon Test on Ablation Study

This section summarizes the performance of different model components. To assess the statistical significance of performance differences, the one-sided Wilcoxon test is applied with a significance level of $\alpha = 0.05$, adjusted with Bonferroni correction. This provides a measure of whether improvements are consistent and not due to random variation.

Table 14: Performance comparison under different model components. The best result and those not statistically different from it are all shown in **bold**. Values are reported as mean \pm standard deviation.

Model Component	Dataset	CA	AD	HI	KDD	JA
	Metrics	RMSE \downarrow			Accuracy \uparrow	
Cross-Dim		0.527 \pm 0.012	0.857 \pm 0.003	0.731 \pm 0.003	0.890 \pm 0.004	0.724 \pm 0.003
Cross-Dim + Cross-Enc		0.508 \pm 0.011	0.857 \pm 0.003	0.733 \pm 0.003	0.904 \pm 0.006	0.727 \pm 0.002
Self-Attention (Simple Concatenation)		0.531 \pm 0.010	0.856 \pm 0.003	0.731 \pm 0.003	0.900 \pm 0.006	0.725 \pm 0.002

Table 14 shows the results of an ablation study comparing different model components. The combination of Cross-Dimensional and Cross-Encoding attention consistently achieves the best or statistically similar performance across all datasets. It gives the lowest RMSE on CA

(0.508) and the highest accuracy on AD, HI, KDD, and JA, showing the benefit of modeling both intra-feature and inter-representation relationships. The Cross-Dimensional only version performs well, particularly on AD and HI, while the Self-Attention with simple concatenation setup shows comparable accuracy on some datasets but falls behind on CA and JA. Overall, the results highlight that combining both attention mechanisms leads to more improvements.

7.5 Performance on Imbalanced Datasets

The following performance metrics are used: AUROC, AUPRC, F1 Score, Recall, and Precision to evaluate unbalanced datasets (AD, KDD, and JA). Since the JA dataset is a multi-class classification problem, the Macro method is applied, where the metric is calculated for each class individually and then averaged across all classes.

Table 15: More performance metrics (AUROC, AUPRC, F1 Score, Recall, and Precision) across different imbalanced datasets (AD, KDD, and JA). Values are reported as mean \pm standard deviation.

Dataset	Model	Accuracy	AUROC	AUPRC	F1 Score	Recall	Precision
AD	XGBoost	0.859 \pm 0.002	0.913 \pm 0.002	0.782 \pm 0.005	0.674 \pm 0.008	0.608 \pm 0.015	0.755 \pm 0.007
AD	FT-Transformer	0.851 \pm 0.002	0.905 \pm 0.002	0.751 \pm 0.005	0.631 \pm 0.08	0.607 \pm 0.163	0.713 \pm 0.092
AD	XDEAT	0.857 \pm 0.002	0.911 \pm 0.002	0.781 \pm 0.004	0.667 \pm 0.034	0.644 \pm 0.111	0.718 \pm 0.071
KDD	XGBoost	0.900 \pm 0.004	0.955 \pm 0.003	0.905 \pm 0.006	0.805 \pm 0.009	0.771 \pm 0.015	0.842 \pm 0.012
KDD	FT-Transformer	0.899 \pm 0.005	0.953 \pm 0.004	0.901 \pm 0.009	0.798 \pm 0.014	0.777 \pm 0.054	0.825 \pm 0.047
KDD	XDEAT	0.902 \pm 0.004	0.958 \pm 0.002	0.912 \pm 0.005	0.795 \pm 0.032	0.790 \pm 0.090	0.817 \pm 0.081
JA	XGBoost	0.701 \pm 0.002	0.855 \pm 0.002	0.595 \pm 0.005	0.516 \pm 0.002	0.516 \pm 0.002	0.584 \pm 0.115
JA	FT-Transformer	0.728 \pm 0.003	0.882 \pm 0.003	0.645 \pm 0.007	0.599 \pm 0.009	0.584 \pm 0.009	0.634 \pm 0.008
JA	XDEAT	0.727 \pm 0.002	0.881 \pm 0.003	0.635 \pm 0.005	0.583 \pm 0.008	0.567 \pm 0.007	0.637 \pm 0.013

As shown in Table 15, the model performs particularly well on the KDD dataset, achieving an AUROC of 0.958 and an AUPRC of 0.912, along with well-balanced F1 Score, Recall, and Precision. This indicates the model’s strong ability to both detect and correctly classify minority class instances. On the AD dataset, performance is also solid, with an AUROC of 0.911 and relatively high precision (0.718), suggesting a low false-positive rate. For the JA dataset, although it involves a multi-class classification task, the AUROC remains robust at 0.881. Despite these strengths, some limitations are observed. The recall values on both the AD (0.644) and JA (0.567) datasets are comparatively lower, indicating that the model may overlook a portion of the minority class samples. These results suggest that while the model is generally effective on imbalanced datasets, especially in binary cases, there is still room to improve its sensitivity to positive instances.

7.6 Training Epoch Analysis

The experimental setup follows the training procedure described in [21], which employs an early stopping mechanism to terminate training when validation performance ceases to improve. To manage computational cost, the maximum number of training epochs is capped at 200, allowing the model to reach its potential without excessive resource usage. As shown in Table 16, the actual number of training epochs varies across datasets, reflecting differences in convergence behavior. For instance, AD and KDD datasets converge relatively quickly (around 55 epochs), while CA and HI require longer training (over 100 epochs). These results suggest that early stopping effectively adapts to dataset complexity, reducing unnecessary training while maintaining competitive performance.

Table 16: Average number of training epochs across datasets, determined by the early stopping mechanism (maximum capped at 200 epochs).

Dataset	CA	AD	HI	KDD	JA
Epochs	139.4	55.133	113.067	54.667	96.933

