Mixture of Attention Yields Accurate Results for Tabular Data

Xuechen Li
Shanghai Warpdrive Technology Co.Ltd
Floor 2, No.57 Boxia Road, Pudong, Shanghai
`lixuechen@warpdriveai.com.cn`

Yupeng Li
Shanghai Warpdrive Technology Co.Ltd
Floor 2, No.57 Boxia Road, Pudong, Shanghai
`liyupeng@warpdriveai.com.cn`

Jian Liu *
Shanghai Warpdrive Technology Co.Ltd
Floor 2, No.57 Boxia Road, Pudong, Shanghai
`liujian@warpdriveai.com.cn`

Xiaolin Jin
Shanghai Warpdrive Technology Co.Ltd
Floor 2, No.57 Boxia Road, Pudong, Shanghai
`jinxiaolin@warpdriveai.com.cn`

Tian Yang
Shanghai Warpdrive Technology Co.Ltd
Floor 2, No.57 Boxia Road, Pudong, Shanghai
`yangtian@warpdriveai.com.cn`

Xin Hu
Shanghai Warpdrive Technology Co.Ltd
Floor 2, No.57 Boxia Road, Pudong, Shanghai
`huxin@warpdriveai.com.cn`

Abstract

Tabular data inherently exhibits significant feature heterogeneity, but existing transformer-based methods lack specialized mechanisms to handle this property. To bridge the gap, we propose MAYA, an encoder-decoder transformer-based framework. In the encoder, we design a Mixture of Attention (MOA) that constructs multiple parallel attention branches and averages the features at each branch, effectively fusing heterogeneous features while limiting parameter growth. Additionally, we employ collaborative learning with a dynamic consistency weight constraint to produce more robust representations. In the decoder stage, cross-attention is utilized to seamlessly integrate tabular data with corresponding label features. This dual-attention mechanism effectively captures both intra-instance and inter-instance interactions. We evaluate the proposed method on a wide range of datasets and compare it with other state-of-the-art transformer-based methods. Extensive experiments demonstrate that our model achieves superior performance among transformer-based methods in both tabular classification and regression tasks.

1 Introduction

As society rapidly advances, tabular data has become one of the most widely used data formats Zhou *et al.* [2025]. Although the information contained in tabular data is highly complex and manifold, due to its significant economic value, there has been increasing interest among researchers in analyzing and studying tabular data, leading to the proposal of various solutions Klambauer *et al.* [2017]; Wang *et al.* [2021]; Popov *et al.* [2020]; Bonet *et al.* [2024]; Arik and Pfister [2021]; Ye *et al.* [2024].

Among these methods, inspired by the amazing success of the transformer architecture in NLP and speech, an increasing number of transformer-like architectures have been introduced to handle
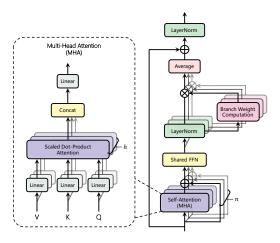
---

*Corresponding author

Figure 1: The overview of MOA block. It employs multiple parallel MHA branches to enhance the richness of feature extraction. To maintain a constant size of hidden states, features are averaged at the branch level. During averaging, each branch's features are weighted by its corresponding branch weight, thereby applying a dynamic consistency constraint. This mechanism facilitates collaborative learning among the branches.

tabular data. AutoInt Song *et al.* [2019] maps both the numerical and categorical features into the same low-dimensional space. Afterwards, a multi-head self-attention neural network with residual connections is proposed to explicitly model the feature interactions in the low-dimensional space. It can be efffciently fit on large-scale raw data in an end-to-end fashion. To investigate the effectiveness of embeddings, Huang et al. Huang *et al.* [2020] propose transforming the embeddings of categorical features into robust contextual representations, and the performance is comparable to that of tree-based methods Chen and Guestrin [2016]; Prokhorenkova *et al.* [2018] on certain datasets. FT-Transformer Gorishniy *et al.* [2021], as a popular transformer-based tabular data model, performs well on multiple datasets and shows the potential for superior performance. Then, ExcelFormer Chen *et al.* [2024] inroduces a semi-permeable attention module, which selectively permits more informative features to gather information from less informative ones. To further improve performance, designing an effective feature interaction module, including intra-instance feature interaction and inter-instance feature interaction Xu *et al.* [2024]; Chen *et al.* [2022], has become a very popular way to enhance model competitiveness. Beyond simple feature interactions, arithmetic feature interactions are also commonly used to enrich feature information and improve feature representation Cheng *et al.* [2024].

A significant limitation of current transformer architectures lies in their inability to effectively process heterogeneous data Chen *et al.* [2024]; Cheng *et al.* [2024]; Chen *et al.* [2023]. While extracting diverse features represents a promising solution to this problem, simply increasing the number of attention heads in Multi-Head Attention mechanism proves inefficient. This approach leads to an expansion of feature dimensions due to the concatenation operation, consequently increasing the parameters quadratically of subsequent Feed-Forward Network (FFN) layers and compromising computational efficiency. Therefore, there exists a critical need to design a transformer architecture that can effectively extract diverse features from tabular data.

In this work, we propose MAYA (**M**ixture of **A**ttention **Y**ields **A**ccurate results for tabular data), a novel encoder-decoder transformer framework that achieves rich feature representations while maintaining parameter efficiency. Specifically, in the encoder, we propose an innovative Mixture of Attention (MOA) structure, as illustrated in Fig.1, which implements parallelized independent attention branches within a single encoder block. This architecture enables effective feature fusion while significantly improving feature diversity for tabular data. Drawing inspiration from collaborative mechanisms Ke *et al.* [2020], the framework employs collaborative policy to weight the outputs of multiple attention branches, ensuring robust and stable feature representation. To further enhance the model's capability, we cascade multiple MOA blocks, enabling the capture of high-level semantic information and significantly improving the encoder's feature representation capacity. The MOA's multi-branch design achieves two key objectives: it extracts diverse features while controlling

dimensionality growth through averaging operations, thereby substantially reducing the parameter burden on downstream FFN layers. This design choice stands in contrast to traditional concatenation approaches, which would lead to quadratic parameter growth in subsequent network components. In the decoder, by integrating an inter-instance attention mechanism and label information, the model achieves more effective and dependable results.

We compared MAYA with a variety of SOTA transformer methods for tabular data, and the superior performance of MAYA is evaluated on several public datasets. Furthermore, our ablation studies also confirmed the effectiveness of the MOA structure of MAYA. Finally, we believe that MAYA can become a new transformer baseline method for tabular data. The main contributions of MAYA are:

- We empirically verify on several public datasets that the proposed MOA structure can effectively obtain diverse features to deal with heterogeneous data in tables.
- We show that the collaborative mechanism applied in branch weight computation of MOA can improve consistency training for tabular data.
- A novel encoder-decoder transformer-based framework is proposed, which employs distinct attention mechanisms at different stages to focus on intra-instance and inter-instance relationships, respectively.

2 Related Work

2.1 Attention Mechanisms

The attention mechanism was introduced to improve the performance of the encoder-decoder model for machine translation Vaswani [2017]. It reveals the relative importance of each component in a sequence compared to the other components. Usually, self-attention and cross-attention are the most widely used in transformer architecture models. Self-attention captures relationships within a single input sequence, and cross-attention captures relationships between elements of different input sequences. Somepalli et al. Somepalli *et al.* [2021] introduce self-attention and inter-sample attention mechanisms for the classification of tabular data. They utilize self-attention to focus on individual features within the same data sample and inter-sample attention to represent the relationships between different data samples of the table. In our approach, we introduce two novel attention mechanisms: the Mixture of Attention (MOA) for self-attention and Inter-instance Attention Incorporating with Labels (IAIL) for cross-attention, to enhance data utilization and feature representation.

2.2 Collaborative Learning

Collaborative learning Dillenbourg [1999] refers to methods and environments in which learners work together on a shared task, with each individual relying on and being responsible to others. In pixel-wise tasks, collaborative policy has been used to train different subset networks and to ensemble the reliable pixels predicted by these networks. As a result, this strategy covers a broad spectrum of pixel-wise tasks without requiring structural adaptation Ke *et al.* [2020]. Here, we use collaborative policy to construct branch weight in MOA block during training, with the weight proportional to the difference between the branch output and the true label. It turns out that this dynamic consistency constraint can provide substantial benefit to enrich feature representations and achieve impressive results for tabular tasks.

3 Methods

In this section, we will provide a detailed introduction to the proposed network MAYA. To present the information more clearly, we will adopt a top-down pattern for our description, starting from the holistic view of the network and gradually delving into its basic blocks.

3.1 Notations

In this work, let's consider supervised learning problems on tabular data. A typical dataset with N instances can be denoted as $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, where x_i represents the features and y_i the label of the i-th instance, respectively. Specifically, x_i contains both numerical features x_i^{num} and categorical
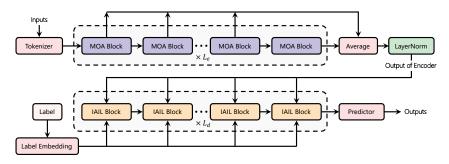
Figure 2: The overall structure of MAYA. It primarily consists of an encoder and a decoder, where the encoder is comprised of MOA blocks (arranged in the upper row) and the decoder is comprised of IAIL blocks (arranged in the lower row). Prior to entering the encoder, the input undergoes processing by the tokenizer. The output of the decoder is then passed through a Predictor to obtain the final prediction results.

features x_i^{cat}, that is to say, $x_i = (x_i^{num}, x_i^{cat})$. To be convenient and without loss of generality, we denote an instance with k features as $x_i \in \mathbb{R}^k$. For the label y_i, three types of tasks are considered, which are binary classification $y_i \in \{0, 1\}$, multi-class classification $y_i \in \{1, \ldots, C\}$ and regression $y_i \in \mathbb{R}$.

3.2 Tokenizer

MAYA is built upon the transformer-like architecture. Consequently, prior to entering the network, instances need to be processed by a tokenizer, which maps the instances from the original feature space into an embedding space. For an instance $x_i \in \mathbb{R}^k$, we define the d-dimension embeddings obtained by tokenizer as z_i:

$$z_i = \text{Tokenizer}(x_i), z_i \in \mathbb{R}^{k \times d} \tag{1}$$

The tokenizer can be implemented in various ways, while in this paper, we primarily utilize the tokenizer of Gorishniy *et al.* [2021], which embeds numerical features through element-wise multiplication and categorical features via a lookup table.

Note that in contrast to Gorishniy *et al.* [2021], we observe that incorporating an activation function subsequent to the tokenizer enhances the predictive performance of the network. We attribute this to the manner in which the tokenizer's weights and biases are initialized: specifically, the use of Kaiming initialization facilitates performance improvement when combined with the ReLU activation function. To further investigate this detail, we will conduct ablation study in Section 4.3.

Before the extraction of features, a common practice is to append the embedding of a special token, i.e. [CLS] token, to z_i:

$$z_i^0 = \text{Stack}[[CLS], z_i], z_i^0 \in \mathbb{R}^{(k+1) \times d} \tag{2}$$

where the subscript "0" denotes the initial input to the network, i.e., the input to the first layer of the entire network.

3.3 Encoder-Decoder Architecture

Our network employs an encoder-decoder framework in its overall design, as demonstrated in Fig.2. The initial input z_i^0 undergoes processing by the encoder, resulting in an output $\hat{z}_i$ (i.e. the processed [CLS] token, which we adopt as the representation of an instance). Subsequently, $\hat{z}_i$ is then passed through the decoder to yield $\tilde{z}_i$. Finally, a predictor is employed to generate the ultimate prediction result p. The whole procedure can be formulated as follows:

$$\hat{z}_i = \text{Encoder}(z_i^0), \hat{z}_i \in \mathbb{R}^d \tag{3}$$

$$\tilde{z}_i = \text{Decoder}(\hat{z}_i), \tilde{z}_i \in \mathbb{R}^d \tag{4}$$

$$p = \text{Predictor}(\tilde{z}_i) \tag{5}$$

4

The predictor is an simple fully-connected layer, and encoder and decoder utilize distinct novel basic blocks, i.e. Mixture of Attention (MOA) block and Inter-instance Attention Incorporated with Labels (IAIL) block, which will be elaborated on in subsequent sections.

3.4 Encoder

In the field of tabular data, model performance often varies significantly across different datasets. To achieve consistently strong performance, we recommend two strategies: deep and wide integration. Deep integration involves the traditional cascading of basic blocks. For wide integration, we introduce a novel MOA block, where multiple individual attentions operate in parallel. Therefore, an encoder consisting of L_e MOA blocks can be formulated as:

$$z_i^\ell = \text{MOA}^\ell(z_i^{\ell-1}), \ell \in [1, L_e] \tag{6}$$

3.4.1 Mixture of Attention (MOA)

An important characteristic of tabular data lies in the heterogeneity of its features, which often renders reliance on a single feature extractor insufficient. For the classical transformer, the Multi-Head Attention (MHA) mechanism enables the model to focus on information from different representation subspaces through various heads Vaswani [2017]. This implies that by increasing the number of heads, we can enhance the transformer's capacity to process diverse features. However, this approach introduces two primary issues: firstly, since the feature subspaces of each head in MHA are concatenated together, increasing the number of heads enlarges the size of attention output hidden states, which substantially increases the number of parameters in the subsequent FFN, leading to higher computational overhead. Secondly, the concatenated features undergo an output projection, which results in feature mixing, thereby diminishing the diversity among multiple head subspaces.

To address these issues, we propose the Mixture of Attention (MOA, as demonstrated in Fig.1). It constructs multiple parallel attention branches based on MHA units and then averages the features at the branch level. This method ensures the diversity of feature extraction while maintaining a constant size of hidden states, thus eliminating the need to increase the parameter count in the FFN. Furthermore, averaging the features acts as a form of regularization, mitigating the risk of overfitting and enhancing robustness, as well as minimizing the impact of potential failures of individual MHAs.

Specifically, the input is replicated multiple times and fed into several completely identical attention branches. These outputs are then processed through a shared FFN (Feed-Forward Network), followed by individual layer normalization. Ultimately, these outputs are averaged and normalized by another layer normalization to serve as the output of a single MOA block. In a rigorous sense, the output z_i^ℓ of ℓ-th MOA block, which comprises n parallel attention branches, is derived as follows:

$$z_i^\ell = \text{LayerNorm}(\frac{1}{n}\sum_{j=1}^{n}\text{Branch}_j(z_i^{\ell-1}) + z_i^{\ell-1}) \tag{7}$$

where $\text{Branch}_j(\cdot) := \text{LayerNorm}_j(\text{FFN}(\text{Attention}_j(\cdot) + (\cdot)))$.

3.4.2 Intra-block Balance: Branch Weight of MOA

The output of a single MOA block is obtained by averaging the results from multiple attention branches, suggesting an equitable contribution of each attention branch's output to the final result. If we regard each attention branch as an individual entity, then this exhibits a paradigm of collaborative learning. On this basis, we can explicitly introduce a dynamic consistency constraint for each branch, leading us to propose a balancing strategy within the MOA block, namely, the utilization of branch weights (Fig.3). Specifically, during training, we attach a shared Predictor (same as in Eq.5) to each attention branch. If the prediction of a particular branch deviates significantly from the ground truth—quantified using Mean Squared Error (MSE) loss for regression tasks and Cross-Entropy (CE) loss for classification tasks—we assign a larger weight to this branch. These weights are computed by applying the softmax function across the losses of all branches. Consequently, branches exhibiting larger deviations receive greater penalty, thereby achieving a balance among the branches. To mitigate significant fluctuations in these branch weights, we apply Exponential Moving Average (EMA) for smoothing purposes. In conclusion, the formulation of branch weight W_B can be articulated as follows:

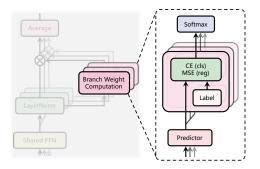$$W_B = \text{EMA}(\text{Softmax}(s)) \tag{8}$$

Figure 3: Computation of branch weights. During training, each branch is appended with the shared Predictor and its loss is computed in relation to the labels. Branches with larger losses are assigned greater weights. This dynamic consistency constraint encourages collaborative learning among the branches.

where $W_B = \{w_j\}_{j=1}^n$ and $s = \{s_j\}_{j=1}^n$. s_j is the supervised loss of Branch_j, that is

$$s_j = \text{L}_{pred}(\text{Predictor}(\text{Branch}_j(z^{\ell-1})), y) \tag{9}$$

where L_{pred} is MSE loss or CE loss.

By applying the branch weight, Eq.7 can be reformulated as

$$z_i^\ell = \text{LayerNorm}(\sum_{j=1}^n w_j \text{Branch}_j(z_i^{\ell-1})) \tag{10}$$

3.4.3 Inter-block Averaging

In pursuit of not only intra-block balance but also inter-block efficacy in feature extraction, we proceed to average the outputs of the stacked L_e MOA blocks and append a LayerNorm layer, thereby yielding the ultimate output of the encoder. Therefore, in summary, combining Eq.3 and Eq.6, the output $\hat{z}_i$ of the encoder is actually obtained through the following calculation:

$$\hat{z}_i = \text{LayerNorm}(\frac{1}{L_e} \sum_{l=1}^{L_e} z_i^l) \tag{11}$$

3.5 Decoder

The decoder consists of a novel inter-instance cross attention module. Given the encoder's proficiency in extracting representations for each individual instance, we innovatively conceptualize a set of instances $\hat{Z} = \{\hat{z}_i\}$ as a sequence, whether it comprises the entire training dataset or a specific batch (i.e. $|\hat{Z}| \leq N$), with each instance functioning as a token within this sequence. For example, in PyTorch, we can unsqueeze a tensor with a shape of $[batchsize, d]$ output by encoder into a tensor of shape $[1, batchsize, d]$. This conceptual shift facilitates the direct utilization of the encoder's output as the input for the inter-instance attention mechanism. On this basis, we introduce a novel foundational module within the decoder, which not only incorporates the inter-instance attention mechanism but also seamlessly integrates label information.

3.5.1 Inter-instance Attention Incorporated with Labels

We refer to the fundamental component of the decoder as IAIL (i.e. Inter-instance Attention Incorporated with Labels), which actually employs a cross-attention mechanism, hence it accommodates multiple inputs, as demonstrated in Fig.4. For clarity, let us designate one input to the ℓ-th IAIL (denoted as IAIL^ℓ) as $\hat{Z}^{\ell-1} = \{\hat{z}_i^{\ell-1}\}$, with its output being $\hat{Z}^\ell = \{\hat{z}_i^\ell\}$; specifically, the input to the first IAIL is defined as $\hat{Z}^0 = \{\hat{z}_i^0\}$. To effectively integrate label information into the processing of IAIL, we introduce a trainable Label Embedding Layer (LEL), which projects labels into an embedding space, formulated as:

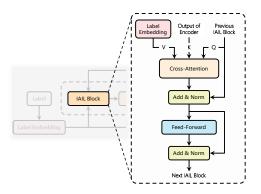$$\hat{y}_i = \text{LEL}(y_i), \hat{y}_i \in \mathbb{R}^d \tag{12}$$

Figure 4: The overview of IAIL block in the decoder.

where, for classification tasks, LEL functions as a lookup table, whereas for regression tasks, it serves as a linear mapper. To maintain consistency with the feature embedding representations, the embeddings of a set of labels are collectively denoted as $\hat{Y} = \{\hat{y}_i\}$.

During the phases of both training and inference, the cross-attention mechanism within IAIL is supplied with distinct inputs, as detailed below:

- Training: for the ℓ-th IAIL, the query is derived from the preceding IAIL's output, while the key is fixed as the raw output obtained from the encoder for instances within the same batch, and the value utilizes the label information corresponding to the instances in the same batch:

$$\hat{Z}_{batch}^{\ell-1} \rightarrow \text{Query}^\ell \tag{13}$$

$$\hat{Z}_{batch}^{0} \rightarrow \text{Key}^\ell \tag{14}$$

$$\hat{Y}_{batch} \rightarrow \text{Value}^\ell \tag{15}$$

  where the subscript "batch" denotes the collective group of instances within a particular batch.

- Inference: during the inference process, we utilize all instances from the training set as the input for both key and value:

$$\hat{z}_i^{\ell-1} \rightarrow \text{Query}^\ell \tag{16}$$

$$\hat{Z}_{train}^{0} \rightarrow \text{Key}^\ell \tag{17}$$

$$\hat{Y}_{train} \rightarrow \text{Value}^\ell \tag{18}$$

  where the subscript "i" signifies the specific i-th test instance, and the subscript "train" indicates the training dataset from which these instances are sourced.

It is noteworthy that for both query and key, we apply identical weights for their respective projections, and assess their similarity through the utilization of the L2 distance instead of scaled dot-product. Consequently, owing to the adoption of the L2 distance, we omit the multi-head mechanism in this context.

4 Experiments

In this section, we will compare the proposed MAYA to SOTA transformer-based methods on several real-world classification and regression datasets to demonstrate its superiority. Furthermore, we embark on a thorough investigation into the attributes of MAYA through ablation studies.

4.1 Experimental Setting

Datasets We have selected 14 datasets from previous works Huang *et al.* [2020]; Somepalli *et al.* [2021]; Gorishniy *et al.* [2021], with detailed information and statistics provided in Table 1. All datasets are partitioned into training, validation, and test sets in a ratio of 64%/16%/20%, respectively, where validation set is utilized for hyper-parameter tuning and early stopping. The preprocessing of all datasets is conducted according to Gorishniy *et al.* [2021].

7

Name	Abbr	Task	Size	#Num	#Cat	#Class	URL
Adult	AD	cls	48842	6	8	2	adult
Bank	BA	cls	45211	7	9	2	bank
Blastchar	BL	cls	7032	3	16	2	blastchar
California Housing	CA	reg	20640	8	0	-	california_housing
Diamonds	DI	reg	53940	6	3	-	diamonds
Helena	HE	cls	65196	27	0	100	helena
Income	IN	cls	48842	6	8	2	income
Jannis	JA	cls	83733	54	0	4	jannis
Otto Group Products	OT	cls	61878	93	0	9	otto_group_products
QSAR Bio	QS	cls	1055	31	10	2	qsar_bio
Seismic Bumps	SE	cls	2584	14	4	2	seismic_bumps
Shrutime	SH	reg	10000	4	6	-	shrutime
Spambase	SP	cls	4601	57	0	2	spambase
Volume	VO	reg	50993	53	0	-	volume

Table 1: The details of datasets. "#Num" and "#Cat" denote the number of numerical and categorical features, respectively.

Evaluation   We primarily adhere to the settings of Gorishniy *et al.* [2021] for our evaluation. Each dataset undergoes experimentation with 15 random seeds, and the average results on the test set are reported. For classification tasks, we employ accuracy as the evaluation metric; for regression tasks, we report the Root Mean Squared Error (RMSE). Besides, we also report the average rank (lower is better) of each method across all datasets to reflect the overall performance of a particular method.

Comparison Methods   The network proposed in this paper will be benchmarked against several SOTA transformer-based methods, including AutoInt (AT) Song *et al.* [2019], TabTransformer (TT) Huang *et al.* [2020], SAINT (SNT) Somepalli *et al.* [2021], FT-Transformer (FTT) Gorishniy *et al.* [2021], ExcelFormer (EF) Chen *et al.* [2024] and AMFormer (AMF) Cheng *et al.* [2024]. All methods are based on their officially released implementations.

Implementation Details   Regarding the comparison methods, if results are reported in the respective papers, we directly cite them. Otherwise, hyper-parameter tuning is conducted within the corresponding space (see Appendix 6.1 for the detailed description). In cases where multiple results are available, we opt to cite the best one. For our method, we leverage Optuna Akiba *et al.* [2019] to execute 100 trials to ascertain the best-performed hyper-parameter configuration, which will be utilized across all 15 random seeds. The hyper-parameter tuning for all methods are based on training and validation sets.

4.2   Main Results

The comparison results of MAYA against other transformer-based methods are presented in Table 2. It is evident that MAYA significantly outperforms other methods in terms of the average rank, indicating that the overall performance of MAYA is currently the best among transformer-based models.

4.3   Ablation Studies

In this section, we undertake an exhaustive analysis of the design characteristics of the proposed network, utilizing six diverse datasets: BA/BL/QS/SE for classification and CA/SH for regression.

The Properties of MOA   First, let us explore the impact of some design details of the MOA block. We set the number of parallel branches in MOA block to 1, thereby degrading the attention mechanism within the MOA block to a single MHA. To ensure a fair comparison, we maintain the same number of heads for MHA as that for MOA in MAYA (i.e., num_heads$_{\mathrm{MHA}}$ = num_heads$_{per\_branch}$ $\times$ num_branch). Based on this configuration, we test two settings: the first ensures that the size of hidden states (i.e., hidden_size) in MHA is identical to that in MOA of MAYA, denoted as MHA$_{hidden\_size}$;

Dataset	AI	TT	SNT	FTT	EF	AMF	MAYA (ours)
AD ↑	0.8576	0.8509	0.8600	0.8588	0.8594	0.8594	**0.8632**
BA ↑	0.9077	0.8998	0.9075	0.9095	0.9092	0.9082	**0.9100**
BL ↑	0.7985	0.7775	0.8008	0.7995	**0.8018**	0.7990	0.8003
CA ↓	0.5007	0.5936	0.4680	0.4564	0.4519	0.4626	**0.4373**
DI$_{\times 10^3}$ ↓	0.5372	0.7345	0.5466	0.5334	0.5331	0.5384	**0.5324**
HE ↑	0.3811	0.3589	0.3856	**0.3890**	0.3802	0.3882	0.3831
IN ↑	0.8605	0.8535	0.8661	0.8633	0.8665	0.8625	**0.8681**
JA ↑	0.7178	0.7084	0.7111	**0.7306**	0.7262	0.7281	0.7207
OT ↑	0.8084	0.8019	**0.8120**	0.8082	0.8035	0.8074	0.7984
QS ↑	0.8357	0.8461	0.8452	0.8330	0.8389	**0.8537**	0.8515
SE ↑	**0.9309**	0.9243	0.9259	0.9275	0.9282	0.9253	0.9299
SH ↓	0.3255	0.3464	0.3131	0.3222	0.3197	0.3149	**0.3126**
SP ↑	0.9362	0.9393	0.9294	0.9424	0.9346	0.9435	**0.9425**
VO$_{\times 10^2}$ ↓	**0.3987**	0.4940	0.4171	0.4210	0.4253	0.4097	0.4037
rank	4.5714	6.4286	4.0000	3.5714	3.6429	3.5000	**2.2857**

Table 2: The results of all methods across 14 datasets. The upward arrow (↑) indicates the accuracy for classification tasks (higher is better), while the downward arrow (↓) represents the RMSE for regression tasks (lower is better). The average rank results also follow the principle that lower is better. The scientific notation next to the dataset names indicates the scale of the results. The best result for each dataset is bolded.

the second guarantees that the subspace dimension per head (i.e., head_dim) in MHA matches that in MOA of MAYA, denoted as MHA$_{head\_dim}$. Other parameters related to the model architecture, such as the upscaling factor for the intermediate layer of FFN (i.e., intermediate_factor) and the number of blocks (i.e., num_layers), remain consistent with MAYA, while the other parameters undergo hyper-parameter tuning as outlined in Section 4.1. Details are provided in Appendix 7.1. Both settings enhance feature richness and diversity by increasing the number of heads. In the first setting, compared to MAYA, the subspace dimension per head of the single-branch MHA is reduced, resulting in equivalent overall feature richness but diminished expressiveness per subspace. In the second setting, compared to MAYA, the output hidden states of the single-branch MHA become larger, leading to an increase in the parameter size of the subsequent FFN, thereby significantly augmenting the model size and computational overhead. By compared to these two single-branch MHA configurations, we want to reveal the advantage of MOA block in balancing feature complexity and parameter counts. Additionally, apart from setting the number of parallel branches to 1, we perform tuning on all other parameters, referred to as MHA$_{free}$. We also conduct ablation study on the branch weights within the MOA block. Specifically, we remove all branch weights and tune the hyper-parameters as described in Section 4.1.

All the experimental results mentioned above are presented in Table 3. It is evident that among all the ablation study settings related to MOA, MAYA achieves the best performance, followed by the configuration without branch weight, both of which outperform the other settings of the single-branch MHA. This indicates that, while enriching feature representations, MOA block's approach of averaging features at the branch level not only balances amount of parameters but also enhances the model's predictive capability in a manner akin to feature pooling. Additionally, the utilization of branch weight effectively encourages collaborative learning among branches, further boosting the model's predictive performance. Conversely, the worst results are observed for MHA$_{hidden\_size}$, suggesting that merely increasing the number of heads while reducing the feature dimension of each head's subspace does not improve the model's predictive capability.

The Influence of Activation after Tokenizer We attempt to remove the activation function following the tokenizer, while maintaining all other configurations constant, and perform hyper-parameter tuning according to the protocol outlined in Section 4.1. This configuration is subsequently compared against MAYA that retains the activation function, to ascertain its impact. The results, presented in Table 4, reveal that the incorporation of a ReLU activation function subsequent to tokenizer yields a marked enhancement in the model's predictive accuracy.

Dataset	MHA$_{hidden\_size}$	MHA$_{head\_dim}$	MHA$_{free}$	w/o W_B	MAYA
BA ↑	0.9084 (0.5×)	**0.9101** (7.3×)	0.9092	0.9088	0.9100
BL ↑	0.8000 (0.4×)	0.7995 (10.7×)	0.7969	0.8001	**0.8003**
CA ↓	0.4479 (0.4×)	0.4412 (12.0×)	0.4405	0.4473	**0.4373**
QS ↑	**0.8578** (0.2×)	0.8559 (12.8×)	0.8288	0.8385	0.8515
SE ↑	0.9275 (0.4×)	0.9278 (6.4×)	0.9287	**0.9306**	0.9299
SH ↓	0.3161 (0.5×)	0.3166 (4.2×)	0.3143	0.3154	**0.3126**
rank	3.8333	3.1667	3.3333	3.0000	**1.6667**

Table 3: The results of ablation studies regarding the MOA block. MHA denotes the use of only one attention branch, where the subscript "hidden_size" indicates that the hidden_size of this attention branch is consistent with that of MOA in MAYA. The subscript "head_dim" signifies that the head_dim of this attention branch aligns with that of MOA in MAYA. Other parameters related to the model structure (i.e. num_heads, intermediate_factor, and num_layers) remain consistent with those in MAYA, while the remaining parameters undergo hyper-parameter tuning. The subscript "free" indicates that all parameters are tuned. w/o W_B represents the removal of the branch weight from MOA. The multiples $N\times$ following the results of MHA$_{hidden\_size}$ and MHA$_{head\_dim}$ indicate that the parameter count of the corresponding encoder is N times that of MAYA. The computation details are provided in Appendix 7.2. The best result for each dataset is in bold. ↑ ∼ accuracy (higher is better), ↓ ∼ RMSE (lower is better).

Dataset	**w/o** activation in tokenizer	MAYA
BA ↑	0.9094	**0.9100**
BL ↑	0.7997	**0.8003**
CA ↓	0.4401	**0.4373**
QS ↑	0.8346	**0.8515**
SE ↑	0.9275	**0.9299**
SH ↓	0.3129	**0.3126**

Table 4: The results of ablation study regarding the activation following the tokenizer. The best result for each dataset is in bold. ↑ ∼ accuracy (higher is better), ↓ ∼ RMSE (lower is better).

5   Conclusion

In this paper, we present MAYA, a novel transformer-based encoder-decoder architecture for tabular data. The design of dual-attention mechanism effectively promote the information fusion within and between instances. Additionally, to improve the ability of extracting diverse features, we introduce MOA to sufficiently utilize multiple attention branches. Experiments on several popular public datasets demonstrate that our method performs impressively in processing tabular data. Furthermore, ablation studies investigate the effectiveness of the MOA blocks and activation operations after tokenizer in our model, suggesting that the multi-branch structure of the MOA blocks, along with activation functions subsequent to tokenizer, is essential for improving the performance of the proposed method for tabular tasks.

References

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

Sercan Ö Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 6679–6687, 2021.

David Bonet, Daniel Mas Montserrat, Xavier Giró-i Nieto, and Alexander G Ioannidis. Hyperfast: Instant classification for tabular data. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11114–11123, 2024.

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

Jintai Chen, Kuanlun Liao, Yao Wan, Danny Z Chen, and Jian Wu. Danets: Deep abstract networks for tabular data classification and regression. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3930–3938, 2022.

Kuan-Yu Chen, Ping-Han Chiang, Hsin-Rung Chou, Ting-Wei Chen, and Darby Tien-Hao Chang. Trompt: towards a better deep neural network for tabular data. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

Jintai Chen, Jiahuan Yan, Qiyuan Chen, Danny Z Chen, Jian Wu, and Jimeng Sun. Can a deep learning model be a sure bet for tabular prediction? In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 288–296, 2024.

Yi Cheng, Renjun Hu, Haochao Ying, Xing Shi, Jian Wu, and Wei Lin. Arithmetic feature interaction is necessary for deep tabular learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 11516–11524, 2024.

Pierre Dillenbourg. *Collaborative learning: Cognitive and computational approaches. advances in learning and instruction series.* ERIC, 1999.

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

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings. *arXiv preprint arXiv:2012.06678*, 2020.

Zhanghan Ke, Di Qiu, Kaican Li, Qiong Yan, and Rynson WH Lau. Guided collaborative training for pixel-wise semi-supervised learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16*, pages 429–445. Springer, 2020.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *Advances in neural information processing systems*, 30, 2017.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *International Conference on Learning Representations*, 2020.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

Noam Shazeer. Glu variants improve transformer. *arXiv preprint arXiv:2002.05202*, 2020.

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

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pages 1161–1170, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. Dcn v2: Improved deep & cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the web conference 2021*, pages 1785–1797, 2021.

Chenwei Xu, Yu-Chao Huang, Jerry Yao-Chieh Hu, Weijian Li, Ammar Gilani, Hsi-Sheng Goan, and Han Liu. Bishop: Bi-directional cellular learning for tabular data with generalized sparse modern hopfield model. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.

Han-Jia Ye, Huai-Hong Yin, and De-Chuan Zhan. Modern neighborhood components analysis: A deep tabular baseline two decades later. *arXiv preprint arXiv:2407.03257*, 2024.

Zhi Zhou, Yu-Kun Yang, Lan-Zhe Guo, and Yu-Feng Li. Fully test-time adaptation for tabular data. In *Proceedings of the 39th AAAI conference on Artificial Intelligence*, 2025.

6 Appendix

6.1 Hyper-parameter Tuning Space

Here we provide hyper-parameter tuning spaces used for Optuna tuning for all methods in Section 4.2.

6.1.1 MAYA

The hyper-parameter tuning space for MAYA is presented in Table 5.

6.1.2 AutoInt

The hyper-parameter tuning space for AutoInt is presented in Table 6.

6.1.3 TabTransformer

The hyper-parameter tuning space for TabTransformer is presented in Table 7.

6.1.4 SAINT

The hyper-parameter tuning space for SAINT is presented in Table 8.

6.1.5 FT-Transformer

The hyper-parameter tuning space for FT-Transformer is presented in Table 9.

6.1.6 ExcelFormer

The hyper-parameter tuning space for ExcelFormer is presented in Table 10.

6.1.7 AMFormer

The hyper-parameter tuning space for AMFormer is presented in Table 11.

7 Ablation Studies

7.1 Implementation Details

In this section, we provide implementation details of the ablation studies on the properties of MOA block. The first two experimental settings (i.e., MHA_{hidden_size} and MHA_{head_dim}) ensure that the number of heads in MHA matches that in MOA of MAYA, which is formulated as:

$$\text{num\_head}_{\text{MHA}} = \text{num\_heads}_{per\_branch} \times \text{num\_branch} \tag{19}$$

For the first setting, MHA_{hidden_size}, we ensure that the hidden_size is consistent with MOA. The hidden_size for MHA should be:

$$\text{hidden\_size}_{\text{MHA}} = \text{head\_dim}_{\text{MHA}} \times \text{num\_head}_{\text{MHA}} \tag{20}$$

	Parameter	Distribution
	num_layers	$\text{UniformInt}[1, 16]$
	hidden_size	$\text{UniformInt}[64, 256]$
	num_heads	$\text{Categorical}[1, 4, 8, 32]$
	num_branch	$\text{UniformInt}[2, 8]$
encoder	intermediate_factor	$\text{Categorical}[0.8, 1, 1.3, 2]$
	dropout	$\text{Uniform}[0, 0.3]$
	add_act	$\text{Categorical}[\text{true}, \text{false}]$
	if_bias	$\text{Categorical}[\text{true}, \text{false}]$
	act_type	$\text{Categorical}[\text{relu}, \text{prelu}]$
	num_decoder_layers	$\text{UniformInt}[1, 16]$
	decoder_intermediate_factor	$\text{Categorical}[0.8, 1, 1.3, 2]$
decoder	decoder_dropout	$\text{Uniform}[0, 0.3]$
	decoder_if_bias	$\text{Categorical}[\text{true}, \text{false}]$
	decoder_act_type	$\text{Categorical}[\text{relu}, \text{prelu}]$
	batch_size	$\text{Categorical}[1024, 2048, 4096]$
others	learning_rate	$\text{LogUniform}[1e\text{-}5, 5e\text{-}3]$
	weight_decay	$\text{Uniform}[0, 0.3]$

Table 5: Hyper-parameter tuning space for MAYA.

Parameter	Distribution
n_layers	$\text{UniformInt}[1, 6]$
d_token	$\text{UniformInt}[8, 64]$
residual_dropout	$\{0, \text{Uniform}[0, 0.2]\}$
attention_dropout	$\{0, \text{Uniform}[0, 0.5]\}$
learning_rate	$\text{LogUniform}[1e\text{-}5, 1e\text{-}3]$
weight_decay	$\text{LogUniform}[1e\text{-}6, 1e\text{-}3]$

Table 6: Hyper-parameter tuning space for AutoInt.

Parameter	Distribution
dim	$\text{Categorical}[32, 64, 128, 256]$
depth	$\text{Categorical}[1, 2, 3, 6, 12]$
heads	$\text{Categorical}[2, 4, 8]$
attn_dropout	$\text{Uniform}[0, 0.5]$
ffn_dropout	$\text{Uniform}[0, 0.5]$
learning_rate	$\text{LogUniform}[1e\text{-}6, 1e\text{-}3]$
weight_decay	$\text{LogUniform}[1e\text{-}6, 1e\text{-}1]$

Table 7: Hyper-parameter tuning space for TabTransformer.

Parameter	Distribution
dim	$\text{Categorical}[16, 32, 64]$
depth	$\text{Categorical}[4, 6]$
heads	$\text{Categorical}[4, 8]$
attn_dropout	$\text{Uniform}[0, 0.5]$
ffn_dropout	$\text{Uniform}[0, 0.5]$
attn_type	$\{\text{colrow}, \text{Categorical}[\text{colrow}, \text{row}, \text{col}]\}$
learning_rate	$\text{LogUniform}[3e\text{-}5, 1e\text{-}3]$
weight_decay	$\{0, \text{LogUniform}[1e\text{-}6, 1e\text{-}4]\}$

Table 8: Hyper-parameter tuning space for SAINT.

Parameter	Distribution
n_layers	UniformInt$[1, 4]$
d_token	UniformInt$[64, 512]$
residual_dropout	$\{0, \text{Uniform}[0, 0.2]\}$
attention_dropout	Uniform$[0, 0.5]$
ffn_dropout	Uniform$[0, 0.5]$
ffn_factor	Uniform$[^2/_3, ^8/_3]$
learning_rate	LogUniform$[1e\text{-}5, 1e\text{-}3]$
weight_decay	LogUniform$[1e\text{-}6, 1e\text{-}3]$

Table 9: Hyper-parameter tuning space for FT-Transformer.

Parameter	Distribution
n_layers	UniformInt$[2, 5]$
d_token	Categorical$[64, 128, 256]$
n_heads	Categorical$[4, 8, 16, 32]$
residual_dropout	Uniform$[0, 0.5]$
mix_type	Categorical$[\text{none}, \text{feat\_mix}, \text{hidden\_mix}]$
learning_rate	LogUniform$[3e\text{-}5, 1e\text{-}3]$
weight_decay	$\{0, \text{LogUniform}[1e\text{-}6, 1e\text{-}3]\}$

Table 10: Hyper-parameter tuning space for ExcelFormer.

Parameter	Distribution
dim	Categorical$[8, 16, 32, 64, 128]$
depth	Categorical$[1, 4]$
attn_dropout	Uniform$[0, 0.5]$
ffn_dropout	Uniform$[0, 0.5]$
num_special_tokens	UniformInt$[1, 4]$
learning_rate	LogUniform$[1e\text{-}5, 1e\text{-}3]$
weight_decay	LogUniform$[1e\text{-}6, 1e\text{-}3]$

Table 11: Hyper-parameter tuning space for AMFormer.

However, there may be cases where the hidden_size cannot be evenly divided by the number of parallel attention branches (i.e., num_branch), resulting in a non-integer value for head_dim$_{\text{MHA}}$ in Eq.20. In such cases, we perform a ceiling operation on head_dim$_{\text{MHA}}$, which may lead to hidden_size$_{\text{MHA}}$ being slightly larger than hidden_size$_{\text{MOA}}$. Please refer to the Table 12 for the specific hidden sizes corresponding to each dataset.

For the second setting, MHA$_{head\_dim}$, we ensure that the subspace dimension for each head, i.e. head_dim$_{\text{MHA}}$, remains identical with MOA, whereby the hidden_size for MHA is still calculated using Eq.20. For the specific hidden sizes corresponding to each dataset, please refer to Table 13.

7.2 Computation of Parameter Counts in Tabel 3

In Table 3 of the main text, we present a comparison of the parameter counts in the encoder of MHA$_{hidden\_size}$ and MHA$_{head\_dim}$ with that of MAYA. Here, we provide the specific method for the calculation.

In the MOA block, each attention branch constitutes an MHA. The MHA is comprised of four linear layers, namely query, key, value, and output projection. The weights of each linear layer form a square matrix of dimensions hidden_size $\times$ hidden_size. Therefore, the number of parameters in the attention mechanism can be calculated as follows:

$$P_{attn} = \text{num\_branch} \times (4 \times \text{hidden\_size}^2) \tag{21}$$

| | MOA in MAYA | | | MHA$_{hidden\_size}$ | | |
	$\#_{branch}$	$\#_{heads}$	D$_{attn}$	$\#_{heads}$	D$_{head}$	D$_{attn}$
BA	4	4	192	$4 \times 4 = 16$	$192 \div 16 = 12$	$16 \times 12 = 192$
BL	6	8	128	$6 \times 8 = 48$	$\lceil 128 \div 48 \rceil = 3$	$48 \times 3 = 144$
CA	6	1	128	$6 \times 1 = 6$	$\lceil 128 \div 6 \rceil = 22$	$6 \times 22 = 132$
QS	8	1	160	$8 \times 1 = 8$	$160 \div 8 = 20$	$8 \times 20 = 160$
SE	4	1	224	$4 \times 1 = 4$	$224 \div 4 = 56$	$4 \times 56 = 224$
SH	3	4	224	$3 \times 4 = 12$	$\lceil 224 \div 12 \rceil = 19$	$12 \times 19 = 228$

Table 12: Details for MHA$_{hidden\_size}$. Notations: $\#_{branch} \sim$ num_branch, $\#_{heads} \sim$ num_heads, D$_{attn} \sim$ hidden_size, D$_{head} \sim$ head_dim.

| | MOA in MAYA | | | | MHA$_{head\_dim}$ | | |
	$\#_{branch}$	$\#_{heads}$	D$_{attn}$	D$_{head}$	$\#_{heads}$	D$_{head}$	D$_{attn}$
BA	4	4	192	$192 \div 4 = 48$	$4 \times 4 = 16$	48	$16 \times 48 = 768$
BL	6	8	128	$128 \div 8 = 16$	$6 \times 8 = 48$	16	$48 \times 16 = 768$
CA	6	1	128	$128 \div 1 = 128$	$6 \times 1 = 6$	128	$6 \times 128 = 768$
QS	8	1	160	$160 \div 1 = 160$	$8 \times 1 = 8$	160	$8 \times 160 = 1280$
SE	4	1	224	$224 \div 1 = 224$	$4 \times 1 = 4$	224	$4 \times 224 = 896$
SH	3	4	224	$224 \div 4 = 56$	$3 \times 4 = 12$	56	$12 \times 56 = 672$

Table 13: Details for MHA$_{head\_dim}$. Notations: $\#_{branch} \sim$ num_branch, $\#_{heads} \sim$ num_heads, D$_{attn} \sim$ hidden_size, D$_{head} \sim$ head_dim.

In the FFN, we employ ReGLU Shazeer [2020], which differs from the original transformer architecture by incorporating three linear layers. Analogous to the naming convention used in LLaMA Touvron *et al.* [2023], we refer to these layers as up, gate, and down projections. Due to the presence of the intermediate_factor (also known as the upscaling factor, see Section 6.1.1), the matrix weights of each linear layer are of dimension intermediate_factor × hidden_size × hidden_size. Consequently, the number of parameters in the FFN can be calculated as follows:

$$P_{ffn} = 3 \times \text{intermediate\_factor} \times \text{hidden\_size}^2 \tag{22}$$

Combining Eq.21 and Eq.22, the number of parameters in an encoder with num_layers of blocks is calculated as follows:

$$P = \text{num\_layers} \times (P_{attn} + P_{ffn}) \tag{23}$$
$$= \text{num\_layers} \times \text{hidden\_size}^2 \tag{24}$$
$$\times (4 \times \text{num\_branch} + 3 \times \text{intermediate\_factor}) \tag{25}$$

According to Eq.23, when calculating the number of parameters for the encoders of MHA$_{hidden\_size}$, MHA$_{head\_dim}$, and MAYA, one simply substitutes the corresponding variables into the equation. Note that the num_layers and intermediate_factor for both MHA and MAYA are always kept consistent. For instance, for the BA dataset (intermediate_factor= 2, num_layers= 15), the number of parameters for the encoders of MHA$_{hidden\_size}$, MHA$_{head\_dim}$, and MAYA are as follows, respectively:

$$P_{\text{MHA}_{hidden\_size}} = 15 \times 192^2 \times (4 \times 1 + 3 \times 2) \tag{26}$$
$$P_{\text{MHA}_{head\_dim}} = 15 \times 768^2 \times (4 \times 1 + 3 \times 2) \tag{27}$$
$$P_{\text{MAYA}} = 15 \times 192^2 \times (4 \times 4 + 3 \times 2) \tag{28}$$

Therefore, the multiples presented in Table 3 are

$$P_{\text{MHA}_{hidden\_size}} / P_{\text{MAYA}} \approx 0.5 \tag{29}$$
$$P_{\text{MHA}_{head\_dim}} / P_{\text{MAYA}} \approx 7.3 \tag{30}$$