DBDW: A Dual-Branch Database Workload Prediction Model

Chao Yang, Tiantian Liu, Fang Peng, and Tianyou Zhu*

Big Data Center, State Grid Corporation of China, Beijing 100052, P.R.China

Corresponding author. E-mail: zhuty94@foxmail.com

Received: Jul. 31, 2025; Accepted: Sep. 01, 2025

Scalable cloud databases allow users to dynamically adjust computational resources based on business needs. However, most current elastic scaling techniques face challenges in making sound decisions due to complex workload variations. Achieving efficient resource allocation requires accurate workload modeling and timely prediction of future workload changes. Therefore, we propose DBDW, an accurate and lightweight model for database workload prediction. DBDW consists of Local Branch and Global branch. The Global branch uses a Mixer architecture to model long-term cloud database workloads at a holistic level. The Local Branch employs multiple components to selectively emphasize local features via a gating strategy. It also uses a channel sparse clustering method to collaboratively represent multichannel information with low computational overhead. Experimental results show DBDW effectively models workload sequences using historical data. Its lowest prediction MAE/MSE for future workloads are 0.5953/0.5971. Furthermore, DBDW demonstrates significantly reduced computational costs: GPU memory usage during training is 304.14 MB , training speed reaches 0.3238 s/epoch , parameter count is 0.34 million, and FLOPs are 4.19 million. This confirms DBDW ensures accurate predictions while greatly reducing overhead, allowing deployment without affecting database operations.

Keywords: Database Workload; Time Series; Mixer; MLP

http://dx.doi.org/10.6180/jase.202607_30.026

1. Introduction

Make sure to format your manuscript in double space format, use 12 point Times New Roman font, and insert line number.

To reduce costs and boost efficiency, a growing number of organizations and individuals now choose cloud databases to store data and conduct business operations [1]. Their scalability and ease of use are key reasons for this popularity [2–5]. Scalable cloud databases let users adjust computing resources on demand, as business needs evolve [1]. However, determining precisely when to scale resources up or down remains an obstacle to wider adoption.

Traditional reactive scaling strategies rely on static threshold rules [6, 7]. They scale resources up when utilization exceeds a preset threshold, and down when it falls below. However, these methods have significant drawbacks:

- Decision lag causes resource provisioning to fail to react promptly to sudden load spikes;
- Over-allocation leads to resource wastage;
- Lack of standardization for threshold configuration frequently causes resource issues.

To overcome these limitations, proactive scaling techniques based on workload prediction have become a research focus. Their core idea is to model historical workload data to predict future resource needs, enabling dynamic alignment of resource supply with demand. Another approach is search-based methods, like Bestconfig [8]. Bestconfig searches for optimal parameters within predeter-

mined ranges based on manually set heuristic rules. However, this method also suffers from two problems: (1) Long search times: To ensure database performance isn't impacted, Bestconfig spends significant time exploring large configuration spaces; (2) Lack of knowledge transfer: When a new tuning request occurs, Bestconfig restarts the search task from scratch, failing to leverage experience from past searches, resulting in inefficient resource use.

Previous research has explored various proactive scaling strategies. Studies [9, 10] achieved precise workload modeling using SQL query statements. However, in highconcurrency production environments, capturing all queries for modeling is impractical and burdens the database. Selecting only a subset of queries introduces another challenge: ensuring the chosen queries sufficiently represent the current workload.Regarding modeling techniques, researchers [10, 11] employed methods like reinforcement learning. While reinforcement learning effectively captures workload patterns across different time periods from historical data, its high experimentation cost hinders building more detailed workload models.

In the field of workload modeling, passive scaling strategies have also been explored. For example, Nguyen et al. [7] periodically collect and analyze fluctuating cloud database metrics, such as CPU and memory, and establish threshold-based rules to guide scaling. For proactive strategies beyond SQL-based methods, DBSeer [12] uses standard log information to predict resource consumption for each transaction type, identify bottlenecks, and estimate throughput. SOPHIA [13] calculates the cost and benefit of reconfiguration steps to determine optimal future reconfigurations. Wang et al. [14] encode streaming query logs into high-dimensional embeddings to capture workload patterns. Non-query-based models like L-PAW [15] use a topsparse auto-encoder with GRU blocks for adaptive prediction. Bi et al. [16] propose an integrated method combining Savitzky-Golay filter, wavelet decomposition, and stochastic configuration networks. Moneyball [17] focuses on reducing latency by predicting pause/resume patterns, while AutoControl [18] detects and mitigates CPU and disk I/O bottlenecks across nodes.

Long-term time series forecasting, a critical task in many domains including cloud database workload prediction, has seen significant advancements. Informer [19] introduces sparse self-attention to reduce computational complexity. Autoformer [20] uses deep decomposition to extract seasonal and trend components. FEDformer [21] leverages frequency-domain information with frequency-enhanced attention. iTransformer [22] treats channels as input sequences to reduce memory and computation. DLinear [23] proposes a simple linear layer approach that outperforms complex models in some tasks. TSMixer [24] uses MLP layers with auxiliary information, while N-HiTS [25] combines hierarchical interpolation and multi-rate sampling. LightTS [26] captures key features from long sequences via interval and continuous sampling, and FreTS [27] utilizes frequency-domain transformation for MLP-based forecasting.

Consequently, developing workload prediction algorithms for cloud databases is a key research goal. These algorithms need to accurately model resource consumption while being efficient enough to support proactive scaling strategies.

Motivated by these challenges, we propose DBDW, a lightweight model for accurately predicting cloud database workloads. Instead of relying on raw SQL queries, DBDW uses historical time-series data (like CPU utilization and memory occupancy) as input, and predicts future values for these same metrics. DBDW features a dual-branch architecture. The global branch employs a Mixer architecture to learn temporal patterns and inter-feature relationships from the input sequence, building a high-level representation of the workload. The local branch focuses on extracting salient short-term features. To better capture important details in long sequences, we propose a gating mechanism for time-domain feature highlighting. Furthermore, we propose a Poisson-based Channel Sparse Aggregation method. This approach enables essential cross-channel communication while maintaining low computational overhead, avoiding unnecessary resource consumption.

In summary, DBDW contributes as follows:

- Our proposed DBDW models cloud database workloads using time-series data like CPU utilization and memory usage. It maintains modeling accuracy while incurring low computational overhead, providing a basis for automatic cloud database scaling decisions.
- We introduce a gating-based method to highlight important temporal features in the input data. This approach effectively identifies key local patterns,helping the model distill essential information more efficiently.
- We propose a sparse channel aggregation method. It facilitates efficient multi-channel interaction while avoiding noise from irrelevant channels and reducing computational costs.
- On the public Alibaba dataset, our model achieves state-of-the-art results compared to existing prediction methods. This demonstrates its efficiency and accuracy.

Here is the arrangement of the remaining sections. Section 2 details our proposed DBDW algorithm and its modules. Section 3 presents the experimental results and analysis. Finally, we discuss our conclusion and future work.

2. Method

This section details the DBDW model and its constituent components. In summary, our model uses historical cloud database time-series data (such as CPU utilization and memory usage) as input. These inputs are processed independently by a Local Branch and a Global Branch. The learned representations are then fused to predict future CPU utilization and memory usage. This enables building an efficient, accurate, and low-overhead database workload model. The Local Branch consists of four key components: the Patch Maker, TMixer, Dynamic Gate, and Sparse CMixer. It extracts detailed local features from the input sequence by dividing it into patches of varying lengths and processing them through these modules. The global branch is built upon TCMixer. By utilizing two layers of TCMixer in series, the Global Branch learns comprehensive global features from the input sequence across both the time dimension and the channel dimension. A schematic diagram of the DBDW model is shown in Fig. 1.

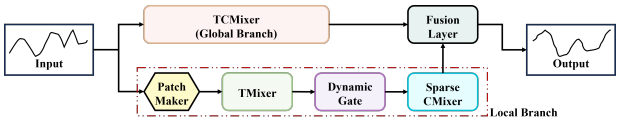

Fig. 1. Architectural details of the DBDW. The DBDW model incorporates both a Local Branch and a Global Branch. The Local Branch is composed of the Patch Maker, TMixer, Dynamic Gate, and Sparse CMixer components

We model the cloud database workload prediction problem as a multivariate time series forecasting task. The problem is formally defined as follows. Given historical workload sequence data $X \in \mathbb{R}^{L\times C}$ representing L past time steps and C channels, our modeling objective is to find a suitable mapping function $f(\cdot)$. This function should produce the predicted future workload sequence $\hat{Y} = f(X) \in \mathbb{R}^{T\times C}$. The goal is to minimize the difference between $\hat{Y}$ and the actual future workload sequence $Y \in \mathbb{R}^{T\times C}$, which T denotes the number of future time steps to predict.

2.1. Patch Macker

When handling sequential data like workload time series, capturing local details effectively is crucial for building robust feature representations. Traditional Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) often rely on fixed-size receptive fields. This limits their ability to adapt to patterns varying across temporal scales. To overcome this limitation and enhance the model's multi-resolution analysis capability, we propose the Patch Maker module. Its primary goal is to explicitly decompose the input sequence into multi-scale patches and map them into a unified latent feature space. This generates richer, more informative representations for subsequent processing stages.

For a given input $X \in \mathbb{R}^{L\times C}$, the innovation of the Patch Maker module lies in its parameterized processing flow. This flow concurrently generates multi-scale feature representations. Specifically, we pre-define a list of K patch sizes. For each patch size P_S, the module performs a non-overlapping sliding window operation along the time dimension. This operation segments the input sequence into consecutive patches, resulting in an intermediate tensor $X_p \in \mathbb{R}^{N_p\times P_s\times C}$. Here, N_p is the number of resulting patches.

To prepare these patches for projection and subsequent feature fusion, the module first
flattens the patch number (N_p) and input channel (C) dimensions of X_p. This yields a tensor shaped $\mathbb{R}^{N^*C\times P_s}$. Then, a projection layer maps the reshaped P_s dimension into a shared latent feature space of dimension d.

Finally, the projected feature tensors obtained for each different patch size P_s are stored in a list. This list ultimately accumulates $\sum_i^K \left\lceil \frac{L}{P_s^i} \right\rceil$ feature tensors. These tensors represent the feature representations of the input sequence at different temporal resolution scales. P_S^i denotes the i-th patch size in the size list.

2.2. TMixer and Dynamic Gate

Effectively capturing key local patterns within multi-scale spatiotemporal features while achieving efficient information condensation remains a core challenge in boosting model performance for time-series modeling. Traditional methods often rely on fixed-weight feature fusion or static temporal modeling mechanisms. These struggle to adaptively focus on critical information at different moments and granularities. This limitation causes important local features to be drowned out by redundant data, restricting the model's ability to represent complex temporal patterns. To address this, our paper introduces the collaborative mechanism of TMixer and Dynamic Gate. As shown in Fig. 2 and Fig. 3, this solution provides an innovative approach via the joint design of temporal feature enhancement and multi-scale gated fusion.

TMixer, as the core temporal feature enhancement com-

ponent, is designed to explicitly mine local dependencies in time-series data through non-linear transformations. Specifically: input multi-scale patch features are first projected by a linear layer for dimension expansion, mapping the original feature space to a higher-dimensional implicit space to enrich representation for interaction. Next, the GeLU activation function captures complex feature interactions using its smooth, probabilistic non-linearity. This effectively filters irrelevant temporal information and strengthens key feature representation. Finally, Dropout regularization helps mitigate overfitting and ensures generalization capability. Essentially, TMixer performs deep mixing of the multi-scale patch features at each timestep. The parameterized chain of linear and non-linear transforms converts original local temporal segments into more discriminative temporal representations, providing high-quality input for subsequent gated fusion.

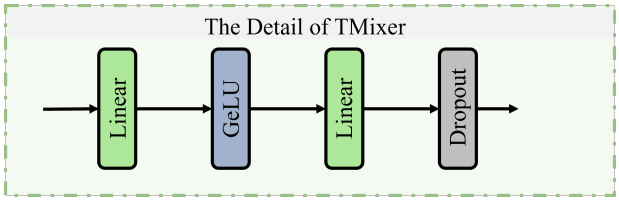

Fig. 2. Architectural details of the TMixer

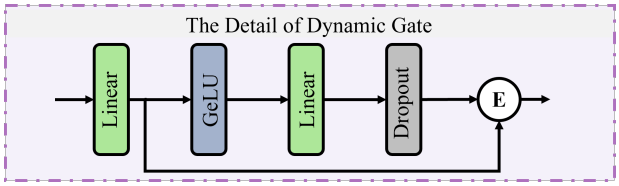

Fig. 3. Architectural details of the Dynamic Gate

The Dynamic Gate module focuses on dynamically weighted fusion of multi-scale features. Its core idea is to adaptively assign weights to features across different time scales and spatial positions using a context-aware gating mechanism, thereby highlighting key local features. Its workflow has three stages: First, the temporally mixed features from TMixer undergo global average pooling. This compresses spatial dimension (i.e., number of multi-scale patches) information into a global context vector, implicitly capturing the statistical distribution of different patch features and providing a global prior for gating weight generation. Second, this context vector is fed into a gating network consisting of a linear layer, GeLU activation, and Softmax function. The linear layer maps the high-dimensional context to a space matching the number of scales. GeLU introduces further non-linearity to enhance feature expressiveness. Softmax finally outputs normalized weights for each scale feature, reflecting their importance to the current input. Third, an einsum operation multiplies the multi-scale features by their corresponding weights and accumulates them, achieving importance-weighted feature fusion. This dynamic gating mechanism allows the model to automatically focus on decisive local temporal segments based on real-time input characteristics.

The synergy between TMixer and Dynamic Gate creates an enhancement-filtering cascade process: Tmixer transforms the original multi-scale patch features into semantically richer temporal representations, providing high-quality candidate features for the gate. The Dynamic Gate then generates dynamic weights based on these candidates and filters key local features via weighted fusion, further enhancing feature discriminability. This design offers dual advantages: First, the joint optimization of temporal mixing and dynamic gating breaks the linear constraints of traditional feature processing, enabling the capture of more complex temporal dependencies. Second, the adaptability of the gating mechanism lets the model dynamically adjust its feature focus strategy according to different input data. This suppresses redundant noise while preserving critical local information, significantly improving the efficiency of information condensation.

2.3. Sparse CMixer

Balancing model expressivity with computational efficiency remains a fundamental challenge in deep neural network design. Conventional fully connected layers facilitate information transfer through dense inter-channel interactions, offering potent nonlinear modeling capabilities. However, this approach incurs substantial computational overhead due to redundant parametric connections, particularly when processing high-dimensional multiscale features. The involvement of numerous irrelevant or noisy channels can significantly impair the model's ability to focus on critical feature representations. To address this limitation, we introduce the Sparse CMixer module-a novel architecture employing a learnable sparse connectivity mechanism that preserves essential feature interactions while drastically reducing redundant computations. This innovation provides an effective solution for efficient high-dimensional feature processing, as visually depicted in Fig. 4.

The core design principle of Sparse CMixer involves dynamically regulating inter-channel information flow via learnable sparse masks. The module first applies a non-linear transformation to input features $X' \in \mathbb{R}^{L\times C}$ using a MLP structure. Specifically, the input undergoes sequen-

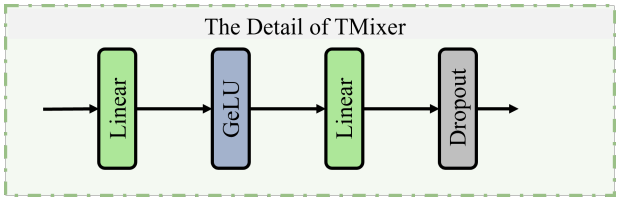

Fig. 4. Architectural details of the TMixer

tial processing through a linear projection layer parameterized by weights $W_1 \in \mathbb{R}^{C\times C\cdot e}$ (with e representing the expansion ratio) and biases b_1, followed by a GeLU activation function, yielding a higher-dimensional hidden representation H as shown in Eq. (1). A subsequent linear layer $W_2 \in \mathbb{R}^{C\cdot exC}$ and biases b_2 then project features back to the original dimensionality H′, calculated from Eq. (2). This parametric transformation maps the input into a semantically enriched latent space, producing high-quality candidate features for subsequent sparse selection.

$$H = \text{GeLU}(W_1 X' + b_1) \tag{1}$$

$$H' = \text{GeLU}(W_2 H + b_2) \tag{2}$$

Diverging fundamentally from traditional dense connectivity, Sparse Cmixer incorporates a learnable sparse mask $M \in \mathbb{R}^C$ that achieves channel-wise sparsity through dynamic binarization. Initialization follows a Bernoulli distribution M from Eq. (3), establishing an initial activation rate of critical channels based on a predefined sparsity level (e.g., 30% activation when sparsity $= 0.3$). During training, mask parameters undergo gradient-based optimization. To ensure sparsity stability, we employ a reinforced binarization strategy: floating-point mask values are processed through a scaled sigmoid function $\text{sigmoid}(M \cdot 5)$, where the scaling factor 5 enhances binarization contrast by driving values toward extremal regions. Thresholding relative to the predefined sparsity level then yields the final binary mask M_{binary} . During inference, M_{binary} remains fixed, and output features Y^0 are computed via element-wise multiplication Eq. (4), enabling selective activation of task-relevant channels while maintaining nonlinear expressive power and significantly reducing computational complexity. Y^0 means the output of Sparse CMixer.

$$M \sim \text{Bernoulli}(1 - \text{sparsity}) \tag{3}$$

$$Y^0 = H' \odot M_{\text{binary}} \tag{4}$$

The Sparse CMixer architecture delivers several compelling advantages. Firstly, its sparse connectivity mechanism substantially diminishes computational load and memory footprint by eliminating unnecessary channel interactions, sustaining high inference efficiency even for large-scale multi-scale features. Secondly, the adaptability of learnable mask parameters allows the sparse pattern to dynamically adjust according to input data-the model selectively activates the most relevant channels per sample, enhancing both the flexibility and discriminative power of feature selection. Finally, intrinsic sparsity inherently provides regularization effects, mitigating overfitting risks associated with redundant channels and thereby improving generalization performance on unseen data.

2.4. TCMixer

Effective modeling of complex workloads necessitates capturing the intrinsic relationship between temporal dynamics and feature characteristics, while simultaneously balancing model expressiveness against training stability. This remains a fundamental challenge in designing core modules. Traditional temporal approaches often exhibit significant limitations: they either emphasize temporal dependency modeling at the expense of feature interactions, or rely on simplistic concatenation of multi-branch outputs, resulting in inefficient information fusion. Consequently, such methods struggle to adapt to the dynamic evolution requirements of multimodal information encountered in complex scenarios.

To address these limitations, we propose TCMixer, a module enabling synergistic information enhancement across both temporal and feature dimensions. TCMixer significantly deepens the representational capability for temporal features while preserving model lightweightness, as quantitatively demonstrated in Fig. 5. The core innovation of TCMixer lies in the construction of a serial processing pathway. This pathway distinctly addresses two critical aspects: localized temporal modeling along the time dimension and highorder semantic fusion along the feature dimension. Crucially, residual connections facilitate lossless information flow and promote stable gradient propagation throughout the network.

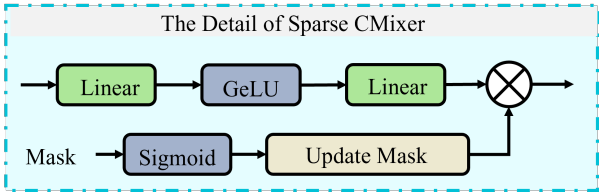

Fig. 5. Architectural details of the Sparse CMixer

To address these limitations, we propose TCMixer, a module enabling synergistic information enhancement

across both temporal and feature dimensions. TCMixer significantly deepens the representational capability for temporal features while preserving model lightweightness, as quantitatively demonstrated in Fig. 5. The core innovation of TCMixer lies in the construction of a serial processing pathway. This pathway distinctly addresses two critical aspects: localized temporal modeling along the time dimension and high-order semantic fusion along the feature dimension. Crucially, residual connections facilitate lossless information flow and promote stable gradient propagation throughout the network.

TCMixer's operation comprises two principal stages. During the Temporal Linear stage, the input feature tensor undergoes flattening, effectively merging time and space dimensions into a unified sequence. This sequence passes through a Temporal Linear processing branch. Batch normalization is applied first to stabilize the distributions of the temporal features. A linear transformation then performs adaptive weighting along the time dimension. Following transposition to recover the original dimension structure, the resulting output is element-wise added to the original input tensor, generating a preliminary residual feature map. Subsequently, the Feature Linear stage leverages this preliminary residual for secondary enhancement within the feature space. Batch normalization again standardizes feature distributions within this branch. Processing then involves a two-stage linear transformation incorporating a central ReLU non-linear activation function and Dropout regularization. This structure induces nonlinear mappings within the feature space, thereby uncovering complex feature interaction patterns.

The final output representation is formed by adding the dual-enhanced feature tensor (enriched temporally and semantically) to the initial residual. This yields a deep representation effectively integrating learned temporal dependencies with high-order feature semantics.

During the Fusion Layer stage, we concatenate the outputs of the two branches. Then, we obtain the final output of the DBDW model through linear projection.

Results and discussion

2.5. Dataset and Baseline}

We use the Alibaba Cluster Trace dataset for experimental evaluation. Specifically, we utilize six time-series database features: "CPU (1 -minute)", "memory (1 -minute)", "disk (1 minute)", "Linux CPU load average (1 -minute)", "Linux CPU load average (5 -minute)", and "Linux CPU load average (15 -minute)". This dataset comprises 187961 data points. We partitioned it into training, validation, and test sets in a 7:1:2 ratio. For comparative evaluation, DBDW is benchmarked against recent SOTA long time-series forecasting models and 2 classic models: iTransformer [22], HDMixer [28], GCformer [29], PatchTST [30], LSTM and GRU. All comparisons were performed using the official implementations provided by the respective authors.

2.6. Implement Setting

MSE, MAE, and MAPE are used as our evaluation metrics. For DBDW, the hidden dimension is set to 128 and the learning rate is 0.001. All models have an input sequence length set to 100 and are evaluated by predicting output sequences of lengths $100, 200, 400$, and 800 . The batch size is 512 . We set the list of patch sizes to [100, 50, 25]. Additionally, Adam is used as the optimizer. All models are implemented in PyTorch and trained/inferenced on an NVIDIA RTX 4090 GPU (24 GB).

2.7. Main Result

Table 1. presents a comprehensive comparison between our proposed DBDW model and six benchmark models (iTransformer, HDMixer, GCformer, PatchTST, LSTM, and GRU) across four prediction lengths (PL $= 100, 200, 400, 800$) and three key metrics (MAE, MSE, MAPE). The evaluation covers 12 distinct test cases (4 PLs $\times$3 metrics), and DBDW achieves first place in 8 cases and second place in the remaining 4 . This consistent top-tier performance clearly demonstrates DBDW's competitiveness in cloud database workload prediction.

When analyzing specific metrics, DBDW stands out prominently in MAE and MSE. For MAE, it reaches the lowest value of 0.5953 at PL $= 100$, outperforming iTransformer (0.5966), HDMixer (0.5956), and even classic models like LSTM (0.6810) by a significant margin. In terms of MSE, DBDW also achieves the best result of 0.5971 at PL $= 100$, which is lower than GCformer (0.6027) and PatchTST (0.5984). Lower MAE and MSE values directly indicate that DBDW minimizes the absolute and squared differences between predicted and actual workload values. This is crucial for cloud database resource scaling, as it means the model rarely produces large prediction errors that could lead to over-provisioning (wasting resources) or under-provisioning (causing performance bottlenecks).

In contrast, DBDW's MAPE performance is slightly less optimal but still competitive. It ranks second only to HDMixer across all prediction lengths. For example, at PL $= 100$, DBDW's MAPE is 11.58%, while HDMixer achieves 11.11%. A higher MAPE for DBDW mainly occurs when forecasting workload metrics with small actual values (e.g., low CPU utilization or memory occupancy). This suggests that DBDW's relative error in predicting lowvalue

Table 1. Multivariate forecasting results of models in terms of MAE/MSE/MAPE. The look-back window size is set to 100, and the prediction lengths (PL) are set to $\{100, 200, 400, 800\}$. The best result is shown in **bold**, followed by underline

Model		Ours	iTransformer	HDmixer	GCformer	PatchTST	LSTM	GRU
PL	Metric							
100	MAE	0.5953	0.5966	0.5956	0.5990	0.5958	0.6810	0.6265
	MSE	0.5971	0.5987	0.6008	0.6027	0.5984	0.7323	0.6461
	MAPE	11.58	11.78	11.11	12.29	11.85	14.61	12.35
200	MAE	0.5959	0.5970	0.5961	0.5980	0.5959	0.6891	0.6106
	MSE	0.5979	0.5994	0.6019	0.6010	0.5986	0.7430	0.6302
	MAPE	11.89	11.98	11.18	12.43	12.03	15.96	12.11
400	MAE	0.5975	0.5980	0.5997	0.6007	0.5977	0.6992	0.6364
	MSE	0.6005	0.6009	0.6069	0.6051	0.6012	0.7662	0.6544
	MAPE	11.88	12.04	11.07	12.56	12.23	13.99	13.06
800	MAE	0.6001	0.6003	0.6061	0.6019	0.6003	0.7291	0.6347
	MSE	0.6040	0.6044	0.6119	0.6069	0.6050	0.8373	0.6549
	MAPE	12.19	12.36	11.59	12.93	12.74	13.91	13.35

targets is marginally higher than HDMixer's, but it still outperforms other models like iTransformer (11.78%) and GRU (12.35%).

To further understand DBDW's superiority, we attribute its strong performance to three core design advantages:

First, the Global-Local Feature Synergy mechanism plays a key role. The Fusion Layer dynamically combines features from the Global Branch (TCMixer-based long-term trend modeling) and Local Branch (multi-scale local detail extraction). This balance allows DBDW to handle both periodic workload patterns (e.g., daily peak hours) and sudden bursty changes (e.g., unexpected query surges)-a capability that single-branch models (like LSTM or GRU) lack.

Second, Dynamic Feature Selection enhances the model's ability to focus on critical information. The Dynamic Gate in the Local Branch adaptively assigns weights to multi-scale patch features using a context-aware gating strategy, effectively suppressing noise from irrelevant time segments. Meanwhile, the Sparse CMixer uses learnable sparse masks to filter out redundant channel information (e.g., less important system metrics), ensuring the model only processes task-relevant features. This reduces computational overhead while improving prediction accuracy.

Third, the dual-branch structure enables effective Error Bound Control. By simultaneously optimizing global trend fitting and local detail capture, DBDW maintains consistently low MAE and MSE across all prediction lengths. Even at the longest PL $= 800$, DBDW's MAE (0.6001) and MSE (0.6040) remain close to its short-term prediction performance-far better than LSTM's MAE of 0.7291 and MSE of 0.8373 . This stability proves DBDW's ability to model complex long-term dependencies in cloud database workloads, which is essential for proactive resource scaling (e.g., planning for weekly or monthly workload fluctuations).

In summary, DBDW's performance across metrics and prediction lengths confirms its reliability for practical cloud database applications. It not only outperforms state-of-the-art models in key error metrics (MAE/MSE) but also maintains competitiveness in relative error (MAPE). This makes DBDW a robust solution for guiding elastic resource allocation, as it can provide accurate workload forecasts while avoiding the severe errors that would disrupt database operations.

2.8. Ablation Study

Table 2. Ablation analysis results of models in terms of MAE/MSE/MAPE/Max Reduct. (%). Max Reduct. ((%) is the abbreviation of "Maximum Reduction (Percentage)". It refers to the largest proportional decrease in value observed for a specific metric when comparing DBDW with a baseline method. Look back window size is set as 100, and prediction length (PL) is set as $\{100, 200, 400, 800\}$. The best result is represented in bold, followed by underline. }

To further examine the contributions of different modules within DBDW, we conducted an ablation study. The results are presented in Table \ref{tab:aba}. Specifically, we designed three variant models:

- WoGate: Replaced TMixer and the Dynamic Gate with an MLP.
- WoSCM: Replaced Sparse CMixer with an MLP.
- WoTCM: Removed the Global Branch, keeping only the Local Branch.

The full model (Ours) significantly outperforms all ablation variants across all forecasting horizons (100/200/400/800), showing clear advantages in MAE,

Table 2. Ablation analysis results of models in terms of MAE/MSE/MAPE/Max Reduct

Model						
		Ours	WoGate	WoSCM	WoTCM	Max Reduct. (%)
PL	Metric					
100	MAE	0.5953	0.6155	0.6008	0.6156	3.298
	MSE	0.5971	0.6255	0.6056	0.6214	4.540
	MAPE	11.58	11.83	11.68	13.28	11.14
200	MAE	0.5959	0.6164	0.6031	0.6168	3.388
	MSE	0.5979	0.6286	0.6046	0.6233	4.884
	MAPE	11.89	12.36	12.77	12.74	6.891
400	MAE	0.5975	0.5992	0.5987	0.6122	2.401
	MSE	0.6005	0.6037	0.6023	0.6194	3.051
	MAPE	11.88	12.20	12.41	12.52	5.112
800	MAE	0.6001	0.6063	0.6097	0.6299	4.699
	MSE	0.6040	0.6104	0.6145	0.6455	6.367
	MAPE	12.19	13.21	12.19	13.15	8.024

Table 3. Hyperparameter sensitivity analysis results of DBDW in terms of MAE, MSE, and MAPE. The look-back window size is set to 100, and the prediction length (PL) is set to 100. The best result is represented in **bold**.

Learning Rate	Metric	Hidden Dim 64	Hidden Dim 128	Hidden Dim 256
0.01	MAE	0.5978	0.5975	0.5979
	MSE	0.5996	0.5995	0.5996
	MAPE	12.15	11.88	12.10
0.001	MAE	0.5956	0.5953	0.5953
	MSE	0.5965	0.5963	0.5965
	MAPE	11.76	11.58	11.59
0.0001	MAE	0.5965	0.5960	0.5962
	MSE	0.5972	0.5970	0.5971
	MAPE	11.77	11.72	11.71

MSE, and MAPE. When prediction length is set as 200 , Ours achieves an MSE of 0.5959 , which is 1.19% lower than the closest ablation variant. Its MAPE of 11.89% is 0.47 percentage points lower than the WoTCM model. This confirms the effectiveness of our model design. The detailed analysis is as follows:

- Replacing TMixer and the Dynamic Gate with an MLP increased MAE by 3.4% at when prediction length is set as 100 . TMixer's temporal mixing capability combined with the Dynamic Gate's adaptive feature selection mechanism are crucial for accurately capturing local dynamic patterns.

- Removing Sparse CMixer led to a 1.6% increase in MAE when prediction length is set as 800. Its core value lies sparse masks during updates, which effectively filters key features in the Local Branch and prevents noise propagation.

- Removing the Global Branch caused MAPE to jump to 13.28%$(+1.7)$ when prediction length is set as 100 . TCMixer constructs global dependencies across timestamps through stacked linear transformations and ReLU activations. This global modeling capability is essential for capturing long-term workload trends.

2.9. Efficiency Analysis

Computational efficiency is critical for database workload predictors. Resource-intensive algorithms may degrade database performance. Motivated by this, we evaluate model
efficiency metrics including GPU memory usage, training time per epoch, parameter count, and FLOPs. Results (Fig. 6 and **??????**) show:

DBDW consumes 304.14 MB GPU memory (3rd lowest). Its training speed is 0.3238 seconds per epoch, marginally slower (0.0345 seconds) than HDMixer (0.2893 s). Crucially, DBDW demonstrates exceptional efficiency in model complexity: it utilizes only 0.34 M parameters (merely 12.43% of HDMixer's) and requires 4.19M FLOPs (just 2.59% of HDMixer's). This confirms DBDW achieves efficient workload modeling and accurate prediction under low computational overhead.

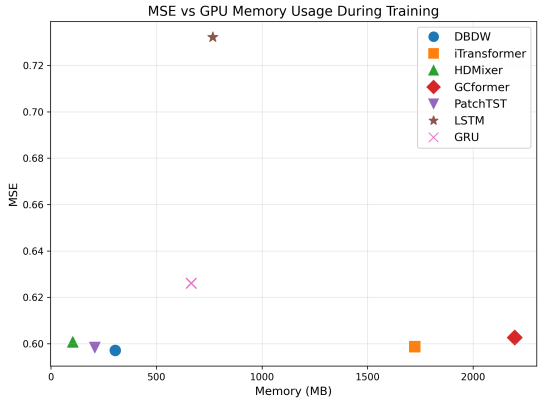

Fig. 6. Comparison of GPU memory usage, prediction length is set as 100

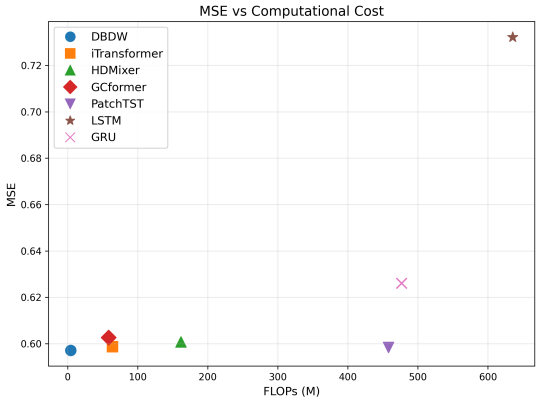

Fig. 9. Comparison of FLOPs, prediction length is set as 100

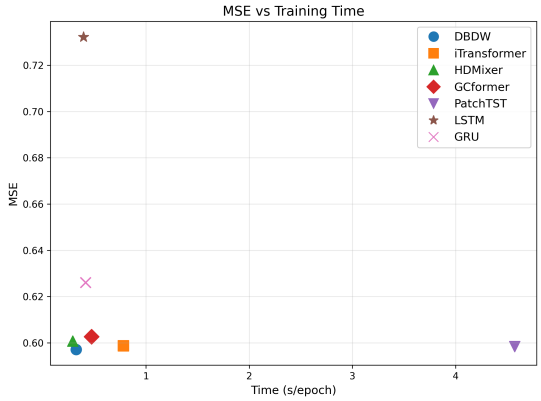

Fig. 7. Comparison of training time per epoch, prediction length is set as 100

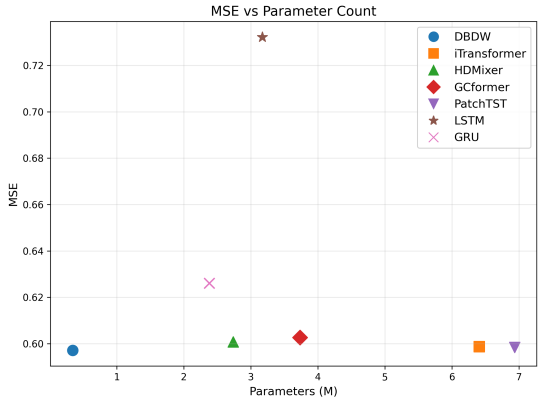

Fig. 8. Comparison of parameter count, prediction length is set as 100

2.10. Hyperparameter Sensitivity Analysis

The results of the hyperparameter sensitivity analysis for the DBDW model show that with the lookback window and prediction length fixed at 100 , different combinations of hidden dimensions $(64, 128, 256)$ and learning rates $(0.1, 0.001, 0.0001)$ have significant impacts on the MAE, MSE, and MAPE metrics. Among them, the learning rate of 0.001 yields better performance in most metrics. For example, when combined with a hidden dimension of 128, it achieves the optimal MAE (0.5953), MSE (0.5963), and MAPE (11.58) within the same learning rate group. Additionally, the hidden dimension of 128 consistently delivers relatively good metrics across different learning rates. These findings indicate that the combination of a learning rate of 0.001 and a hidden dimension of 128 can minimize the prediction error of the model.

3. Conclusion

This paper proposes DBDW, a workload prediction model for cloud databases, featuring both Local Branch and Global Branch. The Local Branch comprises four modules: Patch Maker, TMixer, Dynamic Gate, and Sparse CMixer. It extracts local information from multi-scale patches and employs a gating mechanism where TMixer and Dynamic Gate collaboratively highlight key features. Sparse CMixer then enables efficient cross-channel information exchange with minimal computational overhead. Global Branch is composed of Mixer architecture. Evaluated on the Alibaba Cluster Trace dataset using MAE, MSE, and MAPE metrics across output lengths of $100, 200, 400$, and 800 , DBDW achieved top performance in 8 out of 12 metrics and ranked second in the remaining 4. It attained record-low MAE/MSE values of 0.5953/0.5971. Additionally, DBDW demonstrates practical efficiency with a memory footprint of 304.14 MB during training, 0.3238 seconds per epoch, 0.34 million parameters, and 4.19 million FLOPs. These results confirm DBDW as a lightweight yet highly accurate

workload forecasting solution for cloud database systems.

In future work, we plan to extend the DBDW model in several specific and actionable directions, with a focused enhancement on multi-modal data integration-an area critical for improving prediction robustness in complex cloud database workload scenarios. Beyond the current time-series metrics (e.g., CPU utilization, memory occupancy), we will integrate two key additional data types: structured SQL query logs and granular system event metrics. The SQL query logs will follow a clear format, including fields like query timestamp, type (e.g., SELECT, INSERT), involved table name, execution duration, returned row count, and index usage status; preprocessing will involve classifying queries (transactional vs. analytical) and quantifying complexity via token count or join frequency. System event metrics will cover disk I/O throughput (MB/s), network bandwidth usage (Mbps), cache hit rate (%), connection pool utilization (%), and background task status (e.g., backup, index rebuild), all standardized to align with the existing 1 -minute time granularity. For modal fusion, we will compare two strategies: early fusion, which concatenates preprocessed multi-modal features (with categorical features mapped to continuous spaces via embedding layers and numerical features scaled) into a unified tensor for DBDW's dual branches, and late fusion, which trains separate sub-models for each modality (original DBDW for time-series, BERT for SQL logs, lightweight MLP for system events) and combines their outputs via a learnable weighted sum layer to avoid early heterogeneous data loss.

Additionally, we will address DBDW's current limitation in low-value workload predictions (where MAPE is slightly higher) by designing targeted loss functions that weight errors of low-value samples more heavily or adding adaptive modules to amplify subtle patterns in such data. We will also test the model's generalization across diverse cloud database platforms, including relational (e.g., MySQL) and NoSQL (e.g., MongoDB) systems, using matched multi-modal workload data to verify accuracy without extensive retraining. To support real-time resource adjustment in high-concurrency environments, we will reduce inference latency via knowledge distillation-pruning redundant parameters in TCMixer and Sparse CMixer to train a smaller "student model" with fewer FLOPs. Finally, we will incorporate interpretability modules, such as attention visualization tools, to highlight key predictive features (e.g., specific SQL query types, disk I/O metrics), making DBDW more transparent for database administrators in practical deployment.

4. Acknowledgements

This work is supported by the Self-Established Scientific Project of Big Data Center of State Grid Corporation of China (SGSJ0000YWJS2400086).

References

[1] H. Xu, H. Liu, X. Chen, L. Wang, K. Jin, S. Hou, and Z. Li, (2025) *"Elastic Scaling Method for Multi-tenant Databases Based on Hybrid Workload Prediction Model"* **International Journal of Software and Informatics** ***15***(01): 69–86. DOI: 10.21655/ijsi.1673-7288.00346.

[2] A. Verbitski, A. Gupta, D. Saha, M. Brahmadesam, K. Gupta, R. Mittal, S. Krishnamurthy, S. Maurice, T. Kharatishvili, and X. Bao. "Amazon aurora: Design considerations for high throughput cloud-native relational databases". In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, 1041–1052. DOI: 10.1145/3035918.3056101.

[3] W. Cao, Y. Zhang, X. Yang, F. Li, S. Wang, Q. Hu, X. Cheng, Z. Chen, Z. Liu, J. Fang, et al. "Polardb serverless: A cloud native database for disaggregated data centers". In: *Proceedings of the 2021 International Conference on Management of Data*. 2021, 2477–2489. DOI: 10.1145/3448016.3457560.

[4] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, et al., (2013) *"Spanner: Google's globally distributed database"* **ACM Transactions on Computer Systems (TOCS)** ***31***(3): 1–22. DOI: 10.1145/2491245.

[5] P. Antonopoulos, A. Budovski, C. Diaconu, A. Hernandez Saenz, J. Hu, H. Kodavalla, D. Kossmann, S. Lingam, U. F. Minhas, N. Prakash, et al. "Socrates: The new sql server in the cloud". In: *Proceedings of the 2019 International Conference on Management of Data*. 2019, 1743–1756. DOI: 10.1145/3299869.3314047.

[6] S. Salza and M. Terranova. "Workload modeling for relational database systems". In: *Database Machines: Fourth International Workshop Grand Bahama Island, March 1985*. Springer. 1985, 233–255. DOI: 10.1007/978-1-4612-5144-6_12.

[7] T.-T. Nguyen, Y.-J. Yeom, T. Kim, D.-H. Park, and S. Kim, (2020) *"Horizontal pod autoscaling in kubernetes for elastic container orchestration"* **Sensors** ***20***(16): 4621. DOI: 10.3390/s20164621.

[8] Y. Zhu, J. Liu, M. Guo, Y. Bao, W. Ma, Z. Liu, K. Song, and Y. Yang. "Bestconfig: tapping the performance potential of systems via automatic configuration tuning". In: *Proceedings of the 2017 symposium on cloud computing*. 2017, 338–350. DOI: 10.1145/3127479.3128605.

[9] L. Ma, D. Van Aken, A. Hefny, G. Mezerhane, A. Pavlo, and G. J. Gordon. "Query-based workload forecasting for self-driving database management systems". In: *Proceedings of the 2018 International Conference on Management of Data*. 2018, 631–645. DOI: 10.1145/3183713.3196908.

[10] A. Zafeiropoulos, E. Fotopoulou, N. Filinis, and S. Papavassiliou, (2022) *"Reinforcement learning-assisted autoscaling mechanisms for serverless computing platforms"* **Simulation Modelling Practice and Theory *116***: 102461. DOI: 10.1016/j.simpat.2021.102461.

[11] J. Zhang, K. Zhou, G. Li, Y. Liu, M. Xie, B. Cheng, and J. Xing, (2021) *"CDBTune+: An efficient deep reinforcement learning-based automatic cloud database tuning system"* **The VLDB Journal *30***(6): 959–987. DOI: 10.1007/s00778-021-00670-9.

[12] B. Mozafari, C. Curino, A. Jindal, and S. Madden. "Performance and resource modeling in highly-concurrent OLTP workloads". In: *Proceedings of the 2013 acm sigmod international conference on management of data*. 2013, 301–312. DOI: 10.1145/2463676.2467800y.

[13] A. Mahgoub, P. Wood, A. Medoff, S. Mitra, F. Meyer, S. Chaterji, and S. Bagchi. "{SOPHIA}: Online reconfiguration of clustered {NoSQL} databases for {Time-Varying} workloads". In: *2019 USENIX Annual Technical Conference (USENIX ATC 19)*. 2019, 223–240. DOI: 10.5555/3358807.3358827.

[14] J. Wang, T. Li, A. Wang, X. Liu, L. Chen, J. Chen, J. Liu, J. Wu, F. Li, and Y. Gao, (2023) *"Real-time workload pattern analysis for large-scale cloud databases"* **arXiv preprint arXiv:2307.02626**: DOI: 10.48550/arXiv.2307.02626.

[15] Z. Chen, J. Hu, G. Min, A. Y. Zomaya, and T. El-Ghazawi, (2019) *"Towards accurate prediction for high-dimensional and highly-variable cloud workloads with deep learning"* **IEEE Transactions on Parallel and Distributed Systems *31***(4): 923–934. DOI: 10.1109/TPDS.2019.2953745.

[16] J. Bi, H. Yuan, and M. Zhou, (2019) *"Temporal prediction of multiapplication consolidated workloads in distributed clouds"* **IEEE Transactions on Automation Science and Engineering *16***(4): 1763–1773. DOI: 10.1109/TASE.2019.2895801.

[17] O. Poppe, Q. Guo, W. Lang, P. Arora, M. Oslake, S. Xu, and A. Kalhan, (2022) *"Moneyball: proactive auto-scaling in Microsoft Azure SQL database serverless"* **Proceedings of the VLDB Endowment *15***(6): 1279–1287. DOI: 10.14778/3514061.3514073.

[18] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. "Automated control of multiple virtualized resources". In: *Proceedings of the 4th ACM European conference on Computer systems*. 2009, 13–26. DOI: 10.1145/1519065.1519068.

[19] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. "Informer: Beyond efficient transformer for long sequence time-series forecasting". In: *Proceedings of the AAAI conference on artificial intelligence*. ***35***. 12. 2021, 11106–11115. DOI: 10.1609/aaai.v35i12.17325.

[20] H. Wu, J. Xu, J. Wang, and M. Long, (2021) *"Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting"* **Advances in neural information processing systems *34***: 22419–22430. DOI: 10.48550/arXiv.2106.13008.

[21] T. Zhou, Z. Ma, Q. Wen, X. Wang, L. Sun, and R. Jin. "Fedformer: Frequency enhanced decomposed transformer for long-term series forecasting". In: *International conference on machine learning*. PMLR. 2022, 27268–27286. DOI: 10.48550/arXiv.2201.12740.

[22] Y. Liu, T. Hu, H. Zhang, H. Wu, S. Wang, L. Ma, and M. Long, (2023) *"itransformer: Inverted transformers are effective for time series forecasting"* **arXiv preprint arXiv:2310.06625**: DOI: 10.48550/arXiv.2310.06625.

[23] A. Zeng, M. Chen, L. Zhang, and Q. Xu. "Are transformers effective for time series forecasting?" In: *Proceedings of the AAAI conference on artificial intelligence*. ***37***. 9. 2023, 11121–11128. DOI: 10.1609/aaai.v37i9.26317.

[24] S.-A. Chen, C.-L. Li, N. Yoder, S. O. Arik, and T. Pfister, (2023) *"Tsmixer: An all-mlp architecture for time series forecasting"* **arXiv preprint arXiv:2303.06053**: DOI: 10.48550/arXiv.2303.06053.

[25] C. Challu, K. G. Olivares, B. N. Oreshkin, F. G. Ramirez, M. M. Canseco, and A. Dubrawski. "Nhits: Neural hierarchical interpolation for time series forecasting". In: *Proceedings of the AAAI conference on artificial intelligence*. **37**. 6. 2023, 6989–6997. DOI: 10.1609/aaai.v37i6.25854.

[26] D. Campos, M. Zhang, B. Yang, T. Kieu, C. Guo, and C. S. Jensen, (2023) *"LightTS: Lightweight time series classification with adaptive ensemble distillation"* **Proceedings of the ACM on Management of Data *1***(2): 1–27. DOI: 10.1145/3589316.

[27] K. Yi, Q. Zhang, W. Fan, S. Wang, P. Wang, H. He, N. An, D. Lian, L. Cao, and Z. Niu, (2023) *"Frequency-domain mlps are more effective learners in time series forecasting"* **Advances in Neural Information Processing Systems *36***: 76656–76679. DOI: 10.48550/arXiv.2311.06184.

[28] Q. Huang, L. Shen, R. Zhang, J. Cheng, S. Ding, Z. Zhou, and Y. Wang, (2024) *"Hdmixer: Hierarchical dependency with extendable patch for multivariate time series forecasting"* ***38***(11): 12608–12616. DOI: 10.1609/aaai.v38i11.29155.

[29] Y. Zhao, Z. Ma, T. Zhou, M. Ye, L. Sun, and Y. Qian, (2023) *"Gcformer: an efficient solution for accurate and scalable long-term multivariate time series forecasting"*: 3464–3473. DOI: 10.1145/3583780.3615136.

[30] Y. Nie, N. H. Nguyen, P. Sinthong, and J. Kalagnanam, (2022) *"A time series is worth 64 words: Long-term forecasting with transformers"* **arXiv preprint arXiv:2211.14730**: DOI: 10.48550/arXiv.2211.14730.