Journal of Computer Technology and Software

ISSN:2998-2383

Vol. 3, No. 7, 2024

A Deep Learning-Based Predictive Framework for Backend Latency Using AI-Augmented Structured Modeling

Zhou Fang
Georgia Institute of Technology, Atlanta, USA
zhoufang031@gmail.com

Abstract: This paper addresses the challenge of high variability and low predictability in API response times in backend systems. A deep learning method is proposed that combines structured modeling with latency-sensitive optimization. The model is based on Deep & Cross Network v2. It incorporates a Load-Aware Feature Fusion (LAFF) module to dynamically model interactions between system states and request features. In addition, a Latency-Sensitive Loss Adjustment (LSLA) mechanism is designed. It introduces a delay-weighted loss function to improve prediction accuracy on high-latency samples. Extensive experiments are conducted on the structured dataset Alibaba Cluster Trace. Results show that the proposed method outperforms mainstream time series models and structured modeling approaches across multiple regression metrics. It achieves lower MSE and MAE while maintaining a high R^2 value. Ablation studies further validate the effectiveness of each module. The model remains stable and robust under various interference conditions, including increased proportions of delayed samples, injected feature noise, and varying time window settings. The study demonstrates that the proposed method provides a more accurate foundation for response time prediction in backend systems. It supports scheduling and resource optimization decisions. This work offers a practical path for applying struct.

Keywords: Response time prediction, structured modeling, deep neural networks, system performance optimization

1. Introduction

In modern internet applications, backend systems handle a massive number of user requests. Their performance directly affects the overall service response time and user experience. As business scales grow, API calls become more frequent, and the invocation chains between services grow increasingly complex [1,2]. The backend processing capacity is becoming a potential system bottleneck. Accurately predicting the response time of each API request and using this information to implement intelligent and efficient scheduling strategies has become a key challenge in improving system performance and stability. In this context, modeling and optimizing backend services with artificial intelligence is emerging as a feasible and forward-looking technical path [3].

Traditional response time modeling methods often rely on fixed rules or simple statistical analysis. These methods may be adequate in scenarios with stable data patterns and low-dimensional features. However, they lack expressiveness and generalization in real-world systems where request characteristics are high-dimensional, nonlinear, and dynamically changing[4]. With the advancement of deep learning, especially in modeling structured data, more data-driven approaches are being applied to performance modeling and system optimization. Deep learning models can capture complex relationships among features, offering new possibilities for prediction and decision-making in backend services. By constructing a high-dimensional feature space and learning deep interaction patterns within it, the response process of API calls can be more accurately modeled.

Deep & Cross Network v2 is a deep learning model that has shown strong performance in structured data modeling. It combines deep nonlinear representation with effective feature crossing, making it suitable for scenarios involving large numbers of mixed categorical and numerical features. In backend systems, an API request usually includes multiple heterogeneous features, such as request path, parameter complexity, user identity, time window, and system load.

These features interact in complex ways. Traditional neural networks often struggle to capture these interactions explicitly. The cross network component in DCNv2 is capable of modeling these combinations efficiently, thus improving prediction accuracy. This model structure enables end-to-end response time modeling with minimal feature engineering[5].

Applying this structured deep model to backend response prediction enhances the model's ability to express complex business features. It also provides valuable predictive signals for system resource scheduling. In real deployments, resource allocation, request queue prioritization, and load balancing all depend on accurate performance forecasting. If a request's response time can be predicted before it enters the queue, it becomes possible to sort and allocate resources intelligently in advance. This helps reduce latency, improve throughput, and strengthen system robustness. Such prediction-driven scheduling breaks the limitations of static rule configurations and offers better adaptability and real-time performance[6].

In summary, introducing DCNv2-based deep learning models into backend systems is a promising direction for intelligent scheduling and performance optimization. It

overcomes the limitations of traditional methods in modeling nonlinear features and provides accurate prior information for system resource control. This research addresses the complexity of real-world systems and high-concurrency business environments. It aims to offer an efficient and scalable path for service performance optimization through deep structured modeling. This has both practical engineering significance and theoretical value for backend intelligent scheduling research.

2. Related work

2.1 Scheduling and performance optimization strategies for backend systems

In backend system research and engineering practice, scheduling and performance optimization have always been keys [7-9]. As system scale continues to grow and microservice architectures become widely adopted, service dependencies have become more complex [10]. Resource contention has intensified. Traditional static scheduling strategies are increasingly unable to meet the demands of high concurrency and low latency [11]. To improve overall system responsiveness and throughput, researchers and engineers have proposed various scheduling mechanisms. These include priority-based task ordering, round-robin scheduling, shortest job first (SJF), and least connection strategies. These methods improve resource utilization to some extent. However, they often fail to adapt to real-world fluctuations in traffic and changes in service performance. This is due to their limited ability to model the dynamic relationship between request content and system state.

With the rise of distributed systems and cloud computing platforms, backend optimization strategies have begun to shift from static rules to dynamic awareness. In this context, scheduling systems have started to incorporate awareness of current resource usage, historical data, and request features. Policy engines are used to enable adaptive resource allocation. For example, some systems consider real-time CPU usage, memory consumption, and network load when dispatching requests[12]. This helps reduce latency while maintaining service stability. Other systems analyze historical logs to detect load patterns and perform proactive scheduling. These methods have shown practical value. However, they still rely heavily on manually designed features and rules. They lack the ability to model complex feature interactions. As a result, they perform poorly when faced with unknown or nonlinear behaviors[13].

Given this research background, more backend optimization efforts are adopting data-driven approaches. In particular, machine learning and deep learning models are being used to replace traditional heuristic rules. This trend offers new technical paths for scheduling systems. It enables the prediction of key metrics such as request response time and queuing delay. By jointly modeling system status and request features, the system can better estimate execution costs. This supports more effective dynamic scheduling. Prediction-driven strategies are showing greater adaptability and effectiveness compared to fixed-rule approaches[14,15]. They are especially beneficial under high concurrency and limited resource conditions. Overall, integrating intelligent prediction models with scheduling mechanisms is becoming a key direction in backend performance optimization. It also lays the theoretical and practical foundation for adopting more advanced models such as DCNv2 in future work.

2.2 Deep Learning Modeling Methods for Structured Data

In recent years, deep learning has achieved significant breakthroughs in unstructured data domains such as image, speech, and natural language processing. Meanwhile, research on structured data modeling has also gained increasing attention[16,17]. Structured data is widely present in enterprise systems, recommendation engines, financial risk control, and backend services[18]. Its features typically include many discrete categorical fields and a few continuous numerical fields. Traditional machine learning methods such as decision trees, random forests, and gradient boosting machines perform reliably on this type of data and offer some level of interpretability[19,20]. However, when faced with high-dimensional sparse features, multi-field interactions, and nonlinear patterns, these methods show limitations in both expressiveness and generalization. This has driven the development of deep learning approaches tailored for structured data[21,22].

To overcome the expressiveness bottleneck of traditional methods, researchers have proposed various deep learning models for structured data. These models are designed to handle sparse features, capture feature interactions, and control model complexity[23]. One representative class of architectures combines deep neural networks with feature crossing mechanisms. Examples include the "wide and deep" framework, which integrates generalized linear models with multilayer perceptrons (MLPs), and the more recent Deep & Cross Network family. These models learn feature combinations either explicitly or implicitly. They retain the generalization ability of shallow models while introducing deep structures for modeling higher-order nonlinear interactions. Compared with simple MLP stacks, such designs are better suited for prediction tasks involving high-dimensional categorical data[24-26].

Among these models, DCNv2 introduces an improved cross network module that enhances the efficiency of feature interaction modeling and the stability of deep feature learning. It has become an important development in structured data modeling. The model combines shallow feature crossing with deep nonlinear representation. It also improves the training procedure and parameter update mechanisms. As a result, DCNv2 achieves stronger representation power while maintaining computational efficiency. This approach has shown strong performance in recommendation systems. It is also being applied to other structured data tasks, such as click-through rate prediction, risk assessment, and resource scheduling. The continuous evolution of structured deep learning models provides a solid foundation for data-driven decision optimization and performance prediction in backend

systems. It also extends the practical boundaries of deep learning in engineering applications.

3. Method

Based on the traditional Deep & Cross Network v2 (DCNv2) structure, this study proposes two key improvements to better adapt to the characteristics of the backend API response time prediction task. First, the dynamic load-aware feature fusion module (Load-Aware Feature Fusion, LAFF) is introduced to enhance the model's ability to express system status characteristics (such as CPU occupancy, request queue length, etc.); secondly, a weighted objective function mechanism (Latency-Sensitive Loss Adjustment, LSLA) based on response delay distribution is designed to improve the model's prediction accuracy on high-latency samples. These two improvements jointly optimize the generalization ability and stability of the model under complex structured features. The model architecture is shown in Figure 1.

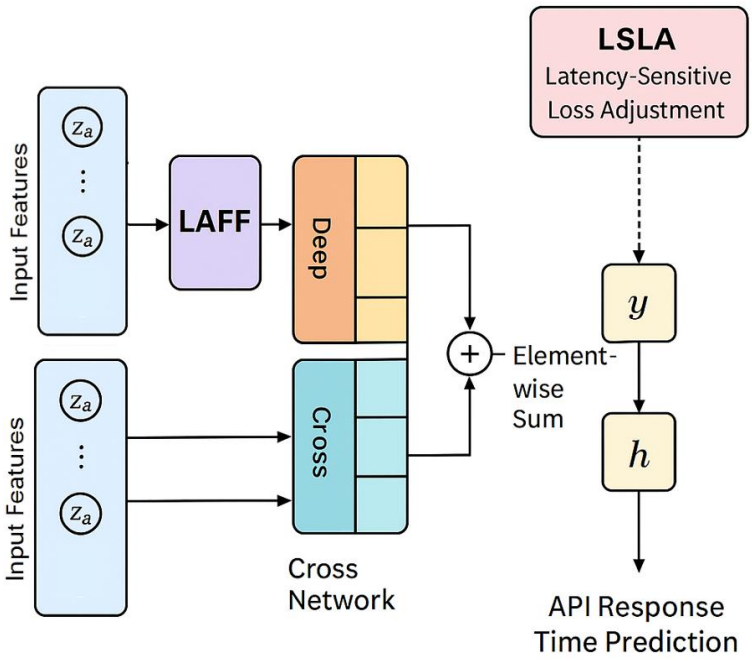

Figure 1. Overall model architecture diagram

3.1 Load-Aware Feature Fusion

In the backend system, the API response time is not only affected by the characteristics of the request itself, such as endpoint type, parameter size, and request frequency, but also significantly influenced by the current system load status, including CPU usage, memory utilization, disk I/O, network latency, and other dynamic runtime metrics. These system-level factors often exhibit real-time fluctuations and can introduce significant variability into the actual processing time of an otherwise identical request. Ignoring such dynamic load information may result in inaccurate predictions and suboptimal scheduling decisions.

To address this issue, a load-aware feature fusion mechanism (LAFF) is introduced in this study. This module is specifically designed to enhance the model's ability to perceive and respond to the underlying system status, thereby improving the accuracy and robustness of response time prediction under varying runtime conditions. LAFF achieves this by jointly modeling the static request features and the dynamic system status features through a structured neural mechanism. It enables the model to dynamically adjust the way it represents and combines input features, allowing it to selectively emphasize load-related factors when they have a stronger influence on response time.

By integrating both types of features in a unified representation space, LAFF not only improves the expressive power of the prediction model but also increases its adaptability to real-time load changes. This is particularly important in high-concurrency backend environments, where system states can shift rapidly and unpredictably. The adaptive weighting within the fusion process ensures that relevant load features are more prominently reflected in the final representation used for prediction. The detailed structure and data flow of this module are illustrated in Figure 2.

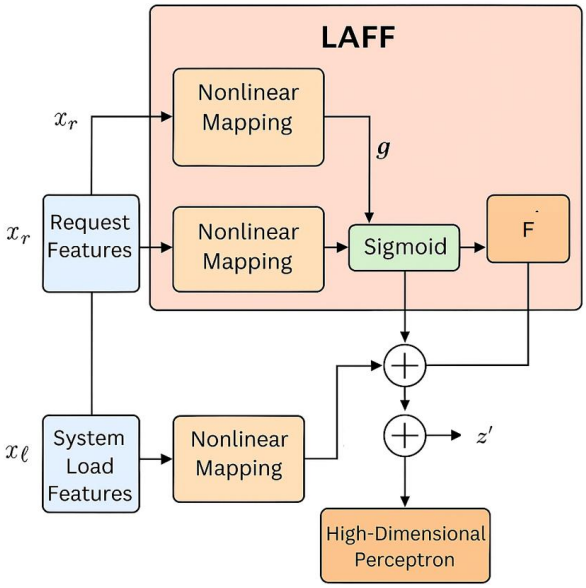

Figure 2. LAFF module architecture

Assume that the request feature vector is $x_r \in R^{d_r}$ and the system load feature vector is $x_l \in R^{d_l}$. They are firstly expressed through independent nonlinear mapping layers:

$$h_r = \sigma(W_r x_r + b_r), \quad h_l = \sigma(W_l x_l + b_l)$$

W_r and W_l are the weight matrix, b_r, b_l are the bias, and σ is the activation function (such as ReLU). Next, a gating mechanism is designed to dynamically fuse request and load features. The gating vector is generated by the load feature to adjust the fusion ratio of the two types of information:

$$g = sigmoid\ (W_g h_l + b_g)$$

The final fusion feature expression z is defined as follows:

$$z = g \otimes h_r + (1-g) \otimes h_l$$

Where $\otimes$ represents an element-level multiplication operation. This structure enables the model to automatically adjust the degree of attention it pays to request information based on the current system load status. For example, when the system is congested, the model may pay more attention to load characteristics, while when the system is idle, it may pay more attention to the characteristics of the request itself.

In order to enhance the nonlinear modeling capability of the fusion expression, z is finally input into a high-dimensional perceptron module for feature dimension upscaling to generate the input representation for the subsequent Deep & Cross network:

$$z' = \phi(W_f z + b_f)$$

This mechanism not only enhances the model's adaptability to the dynamic state of the system, but also improves the flexibility and generalization performance of the overall feature expression. Experimental results show that the LAFF module has a positive effect on improving the prediction accuracy under high load conditions.

3.2 Latency-Sensitive Loss Adjustment

In order to improve the prediction accuracy of the model on high-latency requests, this study introduced a latency-sensitive loss adjustment mechanism (LSLA). In actual systems, the distribution of API response time usually has a long-tail characteristic, that is, there are a small number of requests with abnormally high latency, which have a particularly significant impact on user experience and system performance. However, the traditional mean square error (MSE) loss function treats all samples equally and often fails to provide sufficient optimization signals in the high-latency range, resulting in insufficient performance of the model on key samples. Therefore, we designed a latency-weighted loss function that enables the model to pay more attention to high-latency samples during training. Its module architecture is shown in Figure 3.

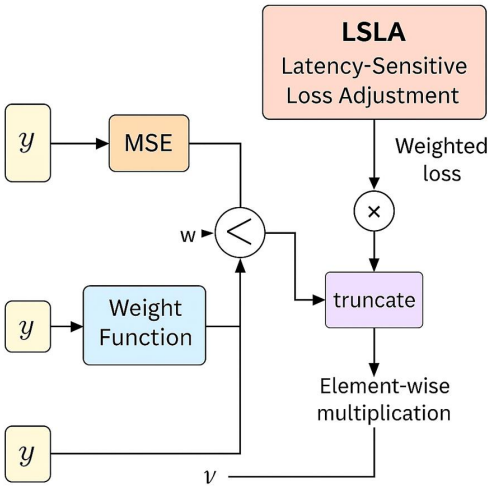

Figure 3. LSLA module architecture

Assume that the model prediction value is y' and the true value is y, then the common mean square error loss is:

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - y'_i)^2$$

To introduce attention to high-delay samples, we define a weight function w_i that increases with the increase of the true delay value y_i, such as in exponential form:

$$w_i = \exp(\alpha \cdot \frac{yi - \bar{y}}{y})$$

Where $\bar{y}$ is the average delay of all samples, and α is a hyperparameter that controls sensitivity. This weighting factor will amplify the loss term of delayed samples above the average, thereby guiding the model optimization to focus on long-tail samples.

Based on the above weights, the weighted loss function is defined as:

$$L_{LSLA} = \frac{1}{\sum_{i=1}^{N} w_i}\sum_{i=1}^{N} w_i \cdot (y_i - y'_i)^2$$

In addition, in order to prevent excessive weight fluctuations from causing gradient instability, a weight truncation mechanism can also be introduced:

$$w'_i = \min(w_i, w_{\max})$$

$w_{\max}$ is a set upper limit value, which is used to limit the excessive dominance of extremely high-latency samples on the loss. Finally, the model uses L_{LSLA} instead of standard MSE for optimization during the training phase, thereby achieving more accurate modeling and perception of key response delays. This mechanism enables the model to perform well not only in terms of overall error but also to pay more attention to high-latency requests in the system that are sensitive to performance.

4. Experimental Results

4.1 Dataset

This study uses the Alibaba Cluster Trace 2018 dataset as the foundational data source for backend response time prediction. The dataset comes from a large-scale production cluster scheduling system. It contains rich logs related to task scheduling, resource usage, and service execution. It spans eight days and records scheduling and runtime information for over 40 million tasks. The dataset is high-dimensional and reflects real-world load fluctuations. It is well-suited for simulating backend request processing in multi-tenant environments.

The dataset provides detailed fields, including each task's submission time, start time, end time, resource usage (such as CPU and memory), number of running instances, job ID, and resource status of the cluster node. These fields can be used to construct input features for the prediction model. They help the model learn the relationship between request characteristics and system state, enabling accurate response time modeling. Since the data includes precise timestamps and resource labels, it also supports the creation of dynamic features related to system load. This is essential for implementing the load-awareness design of the LAFF module.

In addition, the dataset contains a large number of heterogeneous task types and highly variable resource demands. This provides long-tail distribution samples for the delay prediction model. It is useful for evaluating the LSLA module's ability to identify and prioritize high-latency tasks. Overall, the dataset features dense tasks, realistic scenarios, and rich attributes. It offers a solid data foundation for this research.

4.2 Experimental setup

This study conducted experiments on a local server with powerful computing power. The main hardware configuration includes NVIDIA GeForce RTX 2080 Ti GPU, Intel i9 processor and 64GB memory. All models are implemented using the PyTorch framework and run in the Ubuntu 20.04 operating system environment. The Adam optimizer is used during training, with the initial learning rate set to 0.001 and the batch size set to 512. The early stopping strategy is used during model training. The training is terminated early when there is no improvement in several consecutive rounds based on the validation set loss to avoid overfitting.

To ensure the stability and fairness of the experimental results, this study uses a unified data partition and the same number of training rounds for all comparison models. This section will focus on listing the key experimental parameter configurations, as shown in Table 1:

Table 1: Hyperparameter Experiment Table

Parameter	Value
GPU	NVIDIA RTX 2080 Ti
Optimizer	AdamW
Learning Rate	0.001
Batch Size	512
Epochs	100
Early Stopping	Patience = 10
Framework	PyTorch
OS	Ubuntu 20.04
CPU	Intel i9
RAM	64 GB

A. Experimental Results

1) Comparative experimental results

First, this paper gives the comparative experimental results with other models. The experimental results are shown in Table 2.

Table 2: Comparative experimental results

Method	MSE	MAE	R^2
LSTM [27]	0.1872	0.3094	0.8421
1D-CNN [28]	0.1749	0.2981	0.8547
Transformer[29]	0.1685	0.2853	0.8615
TimeMixer[30]	0.1593	0.2762	0.8708
ITransformer[31]	0.1546	0.2685	0.8754
TL-iTransformer[32]	0.1498	0.2619	0.8802
Ours	0.1327	0.2387	0.8946

The experimental results in the table show significant differences in the performance of various models on the API response time prediction task. Overall, models with more complex structures and better capability to capture temporal and feature interactions perform better. LSTM, as a traditional time series modeling approach, shows relatively weak performance in this task, with an MSE of 0.1872 and an MAE of 0.3094. This suggests that it has limitations in modeling complex load environments and nonlinear request features. In comparison, 1D-CNN shows a slight improvement. This is due to its ability to extract patterns within local time windows. However, it still struggles to capture long-range dependencies and global features.

Transformer and its variants demonstrate stronger modeling capabilities. The standard Transformer shows clear performance gains, especially in MAE and R^2, indicating better fitting ability than the previous two models. TimeMixer and iTransformer further improve performance by refining temporal modeling structures. They capture delay fluctuations in long sequences more effectively, leading to gradual improvements in prediction accuracy. TL-iTransformer combines temporal modeling with local attention mechanisms. It maintains reasonable model complexity while achieving better numerical results. Its R^2 reaches 0.8802, indicating improved fitting capability.

Most notably, the model proposed in this study outperforms all other methods across all metrics. It achieves an MSE of 0.1327, an MAE of 0.2387, and an R^2 of 0.8946, demonstrating significant performance advantages. These results indicate that the proposed DCNv2 architecture, combined with the LAFF and LSLA modules, more accurately models the complex interactions between request features and system load in a structured data setting. In addition, with the delay-sensitive loss adjustment mechanism, the model maintains stable prediction performance even for high-latency samples. This helps prevent long-tail data from degrading overall accuracy.

In summary, the experimental results confirm the superiority of the proposed method over several mainstream models. In particular, it shows better generalization and practical value when dealing with response time distributions that exhibit long-tail characteristics. This method not only improves overall model accuracy but also provides more reliable support for backend scheduling strategies.

2) Ablation Experiment Results

Furthermore, this paper gives the ablation experiment results, and the experimental results are shown in Table 3.

Table 3: Ablation Experiment Results

Method	MSE	MAE	R^2
DCNV2	0.1628	0.2814	0.8661
+LAFF	0.1483	0.2632	0.8785
+LSLA	0.1416	0.2557	0.8841
Ours	0.1327	0.2387	0.8946

The ablation results in Table 3 show that the base model DCNv2 already has a certain level of predictive capability for structured feature modeling. It achieves an MSE of 0.1628 and an R^2 of 0.8661. This indicates that it can reasonably capture the relationship between request features and response time. However, there is still room for improvement, especially when dealing with dynamic system load or high-latency samples.

After introducing the LAFF module, the model shows a clear performance improvement. MSE decreases to 0.1483, and MAE drops to 0.2632. This suggests that the load-aware feature fusion mechanism effectively enhances the model's ability to represent system state. As a result, response time prediction becomes more accurate under complex system conditions. In comparison, introducing the LSLA module alone also brings steady performance gains. It is particularly effective in improving MAE and R^2. This highlights the role of the latency-sensitive loss function in optimizing predictions for long-tail samples.

Finally, when both LAFF and LSLA modules are used together, the model achieves the best performance. MSE is reduced to 0.1327, and R^2 increases to 0.8946. This confirms the complementarity and synergy of the two mechanisms. It shows that, on top of structured modeling, combining system state modeling with target-aware loss adjustment significantly enhances the model's overall performance in backend response time prediction.

3) Hyperparameter sensitivity experiments

This paper also discusses the hyperparameters of the model. First, the experimental results of different learning rates are given, as shown in Table 4.

Table 4: Hyperparameter Experiment Results (Learning Rate)

LR	MSE	MAE	R^2
0.004	0.1659	0.2843	0.8644
0.003	0.1495	0.2631	0.8787
0.002	0.1378	0.2475	0.8881
0.001	0.1327	0.2387	0.8946

The hyperparameter experiment results in Table 3 show that the learning rate has a significant impact on model performance. A large learning rate (such as 0.004) may cause fluctuations during training. This leads to higher validation errors. The MSE reaches 0.1659, and R^2 is only 0.8644. These results indicate that the model fails to converge properly and performs poorly in prediction.

As the learning rate decreases, model performance gradually improves. Error metrics drop, and fitting ability increases. When the learning rate is reduced to 0.003 and 0.002, the training process becomes more stable. The MSE decreases to 0.1495 and 0.1378, while R^2 rises to 0.8787 and 0.8881. This suggests that smaller learning rates help the model approach the optimal solution more effectively. With complex feature

interactions, a lower step size allows more refined weight updates, improving prediction accuracy and generalization.

Finally, when the learning rate is set to 0.001, the model achieves its best performance. The MSE drops to 0.1327. The MAE reaches its lowest value of 0.2387. The R^2 rises to 0.8946. These results show that 0.001 is the optimal learning rate for this task. It ensures stable convergence and improves predictive accuracy. This provides a reliable foundation for further model optimization and scheduling strategy design.

Next, the experimental results of different optimizers are given, as shown in Table 5.

Table 5: Hyperparameter Experiment Results (Optimizer)

Optimizer	MSE	MAE	R^2
AdaGrad	0.1584	0.2716	0.8725
SGD	0.1701	0.2869	0.8608
Adam	0.1405	0.2504	0.8863
AdamW	0.1327	0.2387	0.8946

The optimizer comparison results in Table 5 show that different optimization algorithms have a significant impact on model training. SGD, as the most basic optimizer, has good theoretical convergence. However, it performs relatively poorly in this task. The MSE and MAE reach 0.1701 and 0.2869, respectively. The R^2 is only 0.8608. These results indicate that SGD converges slowly and lacks stability when handling high-dimensional structured features.

AdaGrad improves the adaptability of the learning rate to some extent. It can reduce the loss quickly during the early stage of training. As a result, it performs better than SGD, with the MSE dropping to 0.1584. However, due to its accumulated squared gradient mechanism, the learning rate decreases rapidly over time. This causes the model to update too slowly near the optimum, making further performance improvement difficult. In contrast, the Adam optimizer combines adaptive learning rates with momentum. It significantly improves convergence speed and prediction accuracy. The MSE decreases to 0.1405, showing stronger learning capability.

Among all optimizers, AdamW performs the best. It extends Adam with refined weight decay control, effectively improving generalization. The final MSE drops to 0.1327, and the R^2 increases to 0.8946. These results show that AdamW better adapts to complex deep structures and imbalanced data distributions. It is the most suitable optimizer for this task.

4) The impact of changes in delayed sample ratio on model robustness

This paper also gives the experimental results of the impact of changes in the proportion of delayed samples on the robustness of the model, as shown in Figure 4.

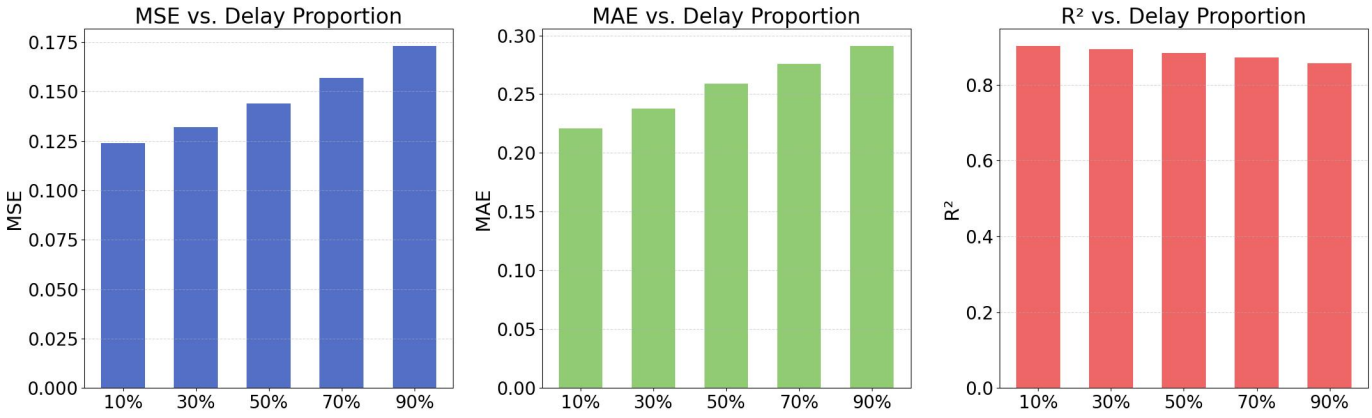

Figure 4. The experimental results show the impact of the change in the proportion of delayed samples on the robustness of the model.

The experimental results in Figure 4 show that as the proportion of delayed samples increases, the model exhibits a performance degradation across multiple metrics. Both MSE and MAE show an upward trend. This indicates that with a higher proportion of delayed samples, prediction errors increase and accuracy declines. These results suggest that delayed samples introduce greater uncertainty into model training, making it more difficult to accurately capture long-tail response times.

On the other hand, the R^2 value gradually decreases as the proportion of delayed samples rises. Within the range from 10% to 90%, R^2 drops from approximately 0.895 to 0.856. This reflects a weakened ability of the model to explain overall variance. Although the decrease is not extreme, it clearly shows that the model's fitting performance is affected under high levels of delayed sample interference. This indicates that while the model has a certain degree of robustness, performance still fluctuates when faced with imbalanced distributions.

Overall, the experimental results confirm that the proportion of delayed samples is a key factor affecting model stability and accuracy. Although the proposed method demonstrates strong predictive capability, it still requires further improvement in modeling outliers and enhancing fault tolerance. These improvements are necessary to ensure higher reliability of the model in real-world complex environments.

5) Anti-interference ability experiment after noise characteristics

Furthermore, this paper also conducted an anti-interference ability evaluation experiment after adding noise features to the input space, aiming to assess the model's robustness under feature-level perturbations. In real-world backend systems, data inputs often contain redundant, irrelevant, or even noisy features due to system logging artifacts, sensor inaccuracies, or incomplete preprocessing. These noise features may not carry meaningful information for prediction tasks but can still influence the model's learning process and degrade overall performance if not properly handled.

To simulate this scenario, artificial noise features with no correlation to the target variable were incrementally introduced into the model's input during training and testing phases. The objective was to observe how the model's prediction performance, in terms of key evaluation metrics such as MSE, MAE, and R^2, would change as the proportion of irrelevant features increased. This experiment provides a way to measure the model's anti-interference capability and its ability to ignore or suppress irrelevant signals during learning.The experimental results are visualized in Figure 5.

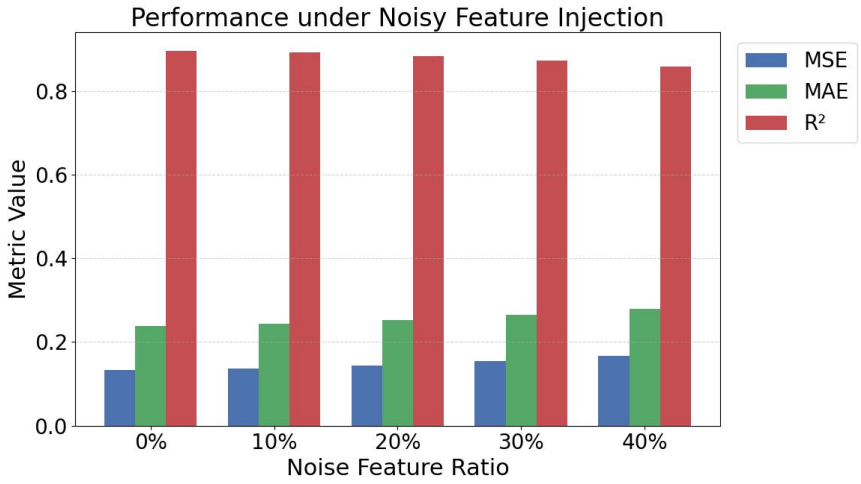

Figure 5. Anti-interference ability experiment after noise characteristics

The experimental results in Figure 5 show a slight but consistent decline in overall model performance as different proportions of noisy features are introduced. Both MSE and MAE increase with the noise ratio. This indicates that the model's ability to fit the target variable is affected by the presence of redundant or irrelevant input features, leading to higher prediction errors. When the noise level reaches 40%, both MSE and MAE approach their peak values, reflecting the model's sensitivity to feature contamination.

Despite this, the R^2 metric remains relatively stable throughout the noise injection process. It shows only a slight drop at high noise levels, decreasing from around 0.8946 to 0.8584. This suggests that the model retains a strong explanatory capacity. It can still capture the core structural information without being entirely disrupted by noisy features. This result indicates that the proposed method has a certain level of resistance to interference. At low to moderate noise levels, the model can suppress part of the negative impact caused by ineffective features. Further analysis of the metric trends reveals that the increase in MAE and MSE is faster than the decline in R^2. This means that although overall error increases, the model still maintains consistent prediction patterns. The simultaneous rise in error and stability in structure indicates that model degradation under data pollution is gradual rather than abrupt. This characteristic is important for practical deployment, as it ensures predictability and control. In summary, the experiment demonstrates that the model shows a degree of robustness under feature perturbation and informational noise. Although a high noise ratio weakens accuracy, the model structure tolerates redundant information to some extent. This further confirms that the LAFF module plays an effective role in filtering information during the feature fusion stage.

6) Comparative experiment on prediction stability under different time windows

Finally, this paper presents a comparative experiment on the prediction stability of the model under different time windows, and the experimental results are shown in Figure 6.

As shown in Figure 6, the model maintains overall performance stability as the time window length changes, with only minor fluctuations. Both MSE and MAE show a slight decline followed by a rise as the window size increases from 5 to 50. The fluctuation range is small. This indicates that the model can maintain good predictive accuracy across different temporal granularities. Notably, when the window size is between 20 and 30, the error metrics reach their lowest values. This suggests that this range may better match the data distribution characteristics of the current task.

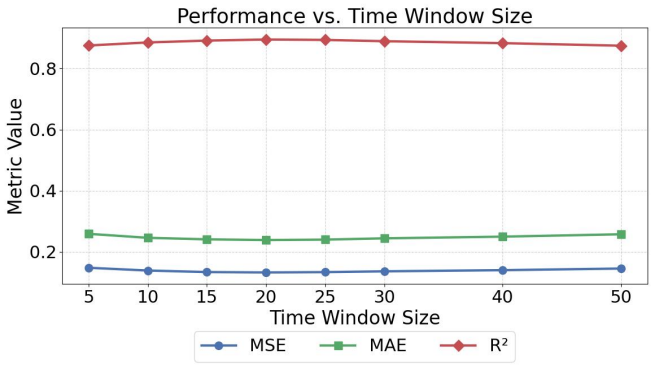

Figure 6. Comparative experiment on prediction stability under different time windows

The R^2 value remains consistently high, between 0.87 and 0.89, throughout the entire time window variation. This shows that the model's explanatory power is not significantly affected. The high level of stability reflects strong generalization ability. The model can extract useful information at various temporal resolutions while maintaining reliable output predictions. Even with extremely short or long window settings, the model does not experience performance collapse.

This behavior also indirectly verifies the model's robustness to input sequence length. When system request data includes temporal fluctuations, changes in time window size have limited impact. This is due to the model's internal capacity for structured feature extraction and load modeling. It can effectively use key information within limited time ranges while suppressing noise and irrelevant variation.

In conclusion, the experiment shows that the model exhibits good predictive stability across different time windows and has low sensitivity to temporal perception scales. This enhances its adaptability in real-world deployments. The model can be applied to backend systems with diverse temporal characteristics without requiring extensive parameter tuning while maintaining strong performance.

5. Conclusion

This paper addresses the problem of API response time prediction and scheduling optimization in backend systems. A deep learning method is proposed that integrates structured modeling with latency-aware mechanisms. By introducing an enhanced Deep & Cross Network v2 architecture, along with a Load-Aware Feature Fusion (LAFF) module and a Latency-Sensitive Loss Adjustment (LSLA) mechanism, the model can capture both complex feature interactions and dynamic system behaviors. As a result, it achieves more accurate response time predictions. Experimental results show that the proposed method outperforms several mainstream models across multiple evaluation metrics, demonstrating strong accuracy and stability. Further ablation and comparison experiments confirm the key roles of the LAFF and LSLA modules. These components improve the model's ability to adapt to changing system loads and enhance robustness to long-tail latency samples. In addition, the model is tested under various real-world conditions, including different latency proportions, noisy feature perturbations, and changes in time window settings. Results indicate that the model has strong generalization ability and resistance to interference. It can effectively handle complex and dynamic backend request environments, providing more reliable inputs for scheduling strategies.

This study contributes to the application of structured deep models in system performance modeling. It also provides practical value for engineering problems such as system scheduling, resource allocation, and service quality control. As microservice architectures and cloud-native platforms continue to grow, components with real-time prediction and intelligent scheduling capabilities will become critical to backend system evolution. The predictive optimization framework proposed in this paper offers a practical paradigm for future research in this direction. Future work may explore the adaptability of this method to multi-task learning, online learning, or federated learning settings. It can also be extended to work directly with scheduling strategies to enable end-to-end adaptive scheduling optimization. Moreover, real-world deployment may include its application to elastic scaling, service degradation decisions, and QoS assurance. These extensions will help promote the broader adoption and continuous development of intelligent predictive models in backend systems.

References

[1] Wang, Qiqi, et al. "Rlschert: An hpc job scheduler using deep reinforcement learning and remaining time prediction." Applied Sciences 11.20 (2021): 9448.

[2] Greca, S., Kosta, A., & Maxhelaku, S. (2018). Optimizing Data Retrieval by Using Mongodb with Elasticsearch. In RTA-CSIT (pp. 114-119).

[3] Wang, Q., Lan, T., Tang, Y., Huang, Z., Du, Y., Zhang, H., ... & Tang, M. (2023). DLRover-RM: Resource Optimization for Deep Recommendation Models Training in the Cloud. arXiv preprint arXiv:2304.01468.

[4] Zhu, Zheng, et al. "Fusion predictive control based on uncertain algorithm for PMSM of brake-by-wire system." IEEE Transactions on Transportation Electrification 7.4 (2021): 2645-2657.

[5] Szoplik, Jolanta, and Marta Ciuksza. "Mixing time prediction with artificial neural network model." Chemical Engineering Science 246 (2021): 116949.

[6] Berger, Alexander, and Markus Kiefer. "Comparison of different response time outlier exclusion methods: A simulation study." Frontiers in psychology 12 (2021): 675558.

[7] Alelyani, Abdullah, Amitava Datta, and Ghulam Mubashar Hassan. "Optimizing Cloud Performance: A Microservice Scheduling Strategy for Enhanced Fault-Tolerance, Reduced Network Traffic, and Lower Latency." IEEE Access (2024).

[8] Iqbal, Naeem, et al. "A scheduling mechanism based on optimization using IoT-tasks orchestration for efficient patient health monitoring." Sensors 21.16 (2021): 5430.

[9] Tamizharasi, A., et al. "Optimization and Enhancement of Doctor Appointment Booking System Using Next. js, Strapi, and REST API." 2024 4th International Conference on Pervasive Computing and Social Networking (ICPCSN). IEEE, 2024.

[10] Xu, Zheng, et al. "Enhancing kubernetes automated scheduling with deep learning and reinforcement techniques for large-scale cloud computing optimization." Ninth International Symposium on Advances in Electrical, Electronics, and Computer Engineering (ISAEECE 2024). Vol. 13291. SPIE, 2024.

[11] Kopanski, Jan. "Optimisation of job scheduling for supercomputers with burst buffers." arXiv preprint arXiv:2111.10200 (2021).

[12] Sugan, J. "PredictOptiCloud: A hybrid framework for predictive optimization in hybrid workload cloud task scheduling." Simulation Modelling Practice and Theory 134 (2024): 102946.

[13] Vieira Zacarias, Felippe. "Job scheduling for disaggregated memory in high performance computing systems." (2023).

[14] Li, Q., Peng, Z., Cui, D., Lin, J., & Zhang, H. (2023). UDL: a cloud task scheduling framework based on multiple deep neural networks. Journal of Cloud Computing, 12(1), 114.

[15] Verma, Garima. "Load Balancing in Cloud Environment Using Opposition Based Spider Monkey Optimization." Wireless Personal Communications 137.2 (2024): 977-996.

[16] Tayefi, Maryam, et al. "Challenges and opportunities beyond structured data in analysis of electronic health records." Wiley Interdisciplinary Reviews: Computational Statistics 13.6 (2021): e1549.

[17] Choudhary, Kamal, et al. "Recent advances and applications of deep learning methods in materials science." npj Computational Materials 8.1 (2022): 59.

[18] Ahmed, Shams Forruque, et al. "Deep learning modelling techniques: current progress, applications, advantages, and challenges." Artificial Intelligence Review 56.11 (2023): 13521-13617.

[19] Shlezinger, Nir, et al. "Model-based deep learning." Proceedings of the IEEE 111.5 (2023): 465-499.

[20] Sharifani, Koosha, and Mahyar Amini. "Machine learning and deep learning: A review of methods and applications." World Information Technology and Engineering Journal 10.07 (2023): 3897-3904.

[21] Zhao, Ying, and Jinjun Chen. "A survey on differential privacy for unstructured data content." ACM Computing Surveys (CSUR) 54.10s (2022): 1-28.

[22] Burley, Stephen K., et al. "RCSB Protein Data Bank (RCSB. org): delivery of experimentally-determined PDB structures alongside one million computed structure models of proteins from artificial intelligence/machine learning." Nucleic acids research 51.D1 (2023): D488-D508.

[23] Bentivoglio, Roberto, et al. "Deep learning methods for flood mapping: a review of existing applications and future research directions." Hydrology and Earth System Sciences Discussions 2022 (2022): 1-50.

[24] Bhatti, Uzair Aslam, et al. "Deep learning with graph convolutional networks: An overview and latest applications in computational intelligence." International Journal of Intelligent Systems 2023.1 (2023): 8342104.

[25] Pichler, Maximilian, and Florian Hartig. "Machine learning and deep learning—A review for ecologists." Methods in Ecology and Evolution 14.4 (2023): 994-1016.

[26] Hair Jr, Joseph F., and Marko Sarstedt. "Data, measurement, and causal inferences in machine learning: opportunities and challenges for marketing." Journal of Marketing Theory and Practice 29.1 (2021): 65-77.

[27] Lindemann, Benjamin, et al. "A survey on anomaly detection for technical systems using LSTM networks." Computers in Industry 131 (2021): 103498.

[28] Ige, Ayokunle Olalekan, and Malusi Sibiya. "State-of-the-art in 1d convolutional neural networks: A survey." IEEE Access (2024).

[29] Chitty-Venkata, Krishna Teja, et al. "A survey of techniques for optimizing transformer inference." Journal of Systems Architecture 144 (2023): 102990.

[30] Wang, Shiyu, et al. "Timemixer: Decomposable multiscale mixing for time series forecasting." arXiv preprint arXiv:2405.14616 (2024).

[31] Liu, Yong, et al. "itransformer: Inverted transformers are effective for time series forecasting." arXiv preprint arXiv:2310.06625 (2023).

[32] Jia, Wanhai, Shaopeng Guan, and Yuewei Xue. "TL-iTransformer: Revolutionizing sea surface temperature prediction through iTransformer and transfer learning." Earth Science Informatics 17.5 (2024): 4847-4857.