Journal of Computer Technology and Software

ISSN:2998-2383

Vol. 3, No. 5, 2024

Intelligent Backend Latency Forecasting with Multi-Scale Convolution and Adaptive Attention Mechanisms

Shaorui Pi[1*]

[1]Carnegie Mellon University, Pittsburgh, USA

*Corresponding author: Shaorui Pi; jesper.shaoruip@gmail.com

Abstract: This paper addresses the problem of backend API latency prediction and proposes a prediction model based on multi-scale convolution and attention mechanisms. The latency sequence is taken as input, and a multi-scale convolution structure is used to extract local and global features at different temporal granularities, capturing both short-term fluctuations and long-term trends. On this basis, an attention mechanism is introduced to highlight the most critical patterns for latency prediction through dynamic weight allocation, effectively reducing redundancy and noise. The convolutional outputs and attention representations are then integrated by a feature fusion layer to generate more discriminative high-level features, which are passed to a regression layer for accurate latency prediction. To validate the model, experiments were conducted with multimodal features including request-level latency, system load, and network fluctuations, and several baseline methods were used for comparison across metrics such as mean squared error, mean absolute error, coefficient of determination, and mean absolute percentage error. The results show that the proposed method outperforms other baseline models in overall performance, better balances local details and global dependencies in latency signals, and significantly improves prediction accuracy and robustness. This study not only enriches modeling approaches for latency prediction but also provides new insights and references for performance assurance and intelligent operations in complex distributed systems.

Keywords: Delay prediction; multi-scale convolution; attention mechanism; distributed system

1. Introduction

In the context of the rapid development of internet applications and distributed architectures, the performance and stability of backend systems have become critical to supporting user experience and business continuity[1, 2]. As a key indicator in service invocation chains, API call latency directly affects response speed and overall system availability. In cloud computing and microservice environments, a single request often triggers multiple cascading service calls, and latency at any stage can be amplified and eventually perceived by users. Therefore, accurate prediction and proactive intervention of API call latency have become essential research topics for backend system optimization and intelligent operations.

With the continuous growth of business scale and concurrent requests, backend systems show high dynamism and complexity. Traditional latency analysis methods often rely on statistical modeling or empirical rules, which are difficult to capture time-varying features and multidimensional dependencies in complex scenarios[3]. Latency is not only related to the load level of a single node but is also influenced by network fluctuations, changes in request patterns, and resource scheduling strategies. These intertwined factors make latency prediction not just a simple time series modeling task but a problem involving spatial structures, contextual information, and nonlinear interactions. Developing methods that can effectively represent these complex features is of great importance for improving the accuracy and robustness of predictions[4].

In recent years, deep learning has shown strong capabilities in sequence modeling and feature extraction, offering new ideas for latency prediction tasks. However, single-scale convolution or single-layer sequence modeling often struggles to balance fine-grained local features and long-term global dependencies. Latency signals usually contain instant jitter, periodic fluctuations, and long-term trends, which require the model to have multi-scale feature-capturing abilities. At the same time, different dimensions of latency features contribute differently to prediction results, making the effective allocation of modeling focus a key direction. Against this background, introducing multi-scale convolution and attention mechanisms can better capture latency features at different levels and highlight critical patterns dynamically, providing more targeted representational power for prediction models[5].

Latency prediction is not only a research challenge but also of direct value in practice[6]. For internet platforms, early awareness of potential latency increases helps adopt resource scheduling, load balancing, or caching strategies before problems worsen, thus ensuring service stability. For cloud and edge computing environments, latency prediction is central to quality assurance and resource optimization[7]. With proper prediction mechanisms, systems can respond proactively rather than reactively to sudden traffic surges and potential bottlenecks, enhancing user experience and reducing operational costs. Such predictive ability is related not only to performance optimization but also to service reliability, energy management, and economic benefits.

In summary, research on backend API call latency prediction based on multi-scale convolution and attention mechanisms addresses both technical demands for handling system complexity and practical needs for intelligent operations and efficient resource management. Exploring the integration of multi-scale feature modeling and dynamic dependency capturing can provide new solutions for latency prediction and establish theoretical and methodological foundations for intelligent evolution of backend systems. This research is not only significant at the academic level but also highly valuable in industrial applications, offering strong support for stable operation and efficient scheduling in large-scale distributed systems.

2. Related Work

In the field of backend system performance modeling and prediction, latency has always been a key focus. Early studies mainly relied on traditional statistical methods and queuing theory models. These methods attempted to model system queuing behavior and service times mathematically, aiming to derive the distribution and variation of latency. However, due to the complexity of distributed architectures and dynamic network environments, purely mathematical modeling often fails to capture nonlinear and high-dimensional dependencies, which limits prediction accuracy and applicability in practice. This stage of research provided a theoretical basis for latency prediction but was insufficient to handle the complex scenarios of modern large-scale systems.

With the development of machine learning, more studies began to adopt regression analysis, tree-based models, and ensemble learning methods to model and predict multidimensional system metrics. These methods improved the ability to fit complex features and achieved good results when the data scale was moderate. However, traditional machine learning models rely heavily on feature engineering. Features such as CPU utilization, memory usage, request size, and concurrency must be manually designed and selected. When system scale is large, feature dimensions are diverse, and dynamics are strong, the limitations of manual feature extraction become clear, making it difficult to support high-accuracy latency prediction.

The introduction of deep learning has provided new approaches for latency prediction. Convolutional neural networks, recurrent neural networks, and attention-based models have gradually been applied to time series and system performance data modeling. Studies have shown that these methods can automatically extract multi-level features from raw data without complex feature engineering, capturing nonlinear patterns and long-term dependencies. Attention-based models in particular can dynamically assign weights and highlight the most critical features for latency prediction, improving both generalization and interpretability. These advances have promoted the shift from static modeling to dynamic and intelligent modeling, pushing latency prediction toward higher accuracy and stronger adaptability.

In further research, multi-scale feature modeling has become an important direction. Latency signals often contain instant fluctuations, periodic variations, and long-term trends, which cannot be fully represented by single-scale modeling. Multi-scale convolution can extract both local and global features under different receptive fields. Combined with attention mechanisms, it can highlight key patterns during feature fusion and reduce interference from redundant information. In recent years, studies have explored combinations of convolutional structures, sequence modeling, and attention mechanisms to achieve more efficient spatiotemporal feature capture. This approach provides new opportunities for backend API latency prediction, offering value in both methodological exploration and industrial applications.

3. Proposed Approach

In this study, the overall framework of the proposed model is centered on a multi-scale convolutional structure to capture the characteristics of delayed signals at different time granularities. Specifically, the input delay sequence is first represented as a vector sequence $X = \{x_1, x_2, .., x_T\}$, where T represents the time step. To extract multi-scale temporal patterns, the model designs convolution operations with different convolution kernel sizes to perform feature extraction in the short and long term. Formally, for a convolution layer with a convolution kernel size of k, its feature representation is:

$$H_i^{(K)} = \sigma(\sum_{i=0}^{k-1} W_i^{(k)} \cdot x_{t-i} + b^{(k)})$$

Where $H_i^{(K)}$ represents the convolution feature at time step t, $\sigma(\cdot)$ is the nonlinear activation function, and $W_i^{(k)}$ and $b^{(k)}$ are the weight and bias parameters, respectively. By splicing the multi-scale convolution results, a more comprehensive time series representation can be obtained:

$$H_t = \bigoplus_{k \in K} H_t^{(k)}$$

Where K represents the set of selected convolution kernel sizes and $\oplus$ represents the concatenation operation. The model architecture is shown in Figure 1.

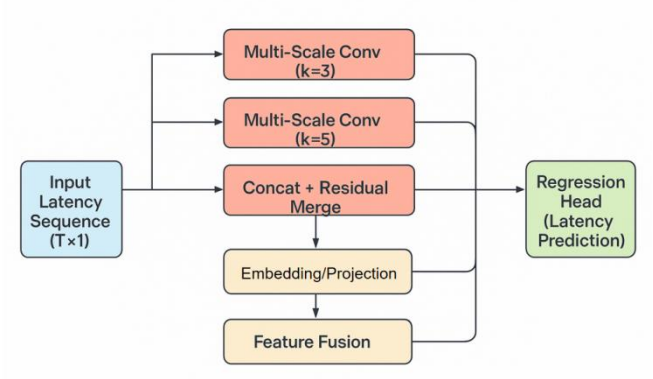

Figure 1. Overall model architecture

After obtaining multi-scale convolutional features, the model further introduces an attention mechanism to improve the expressiveness of key features. The attention module dynamically assigns weights by calculating the correlation

between different time steps. For a given query vector q_i, key vector k_j, and value vector v_j, the attention weight is calculated as follows:

$$a_{ij} = \frac{\exp(q_i \cdot k_j / \sqrt{d_k})}{\sum_{j=1}^{T} \exp(q_i \cdot k_j / \sqrt{d_k})}$$

Where d_k is the dimension of the key vector, which is used for normalization. The value vector is then weighted and summed based on the attention weight:

$$z_i = \sum_{j=1}^{T} a_{ij} v_j$$

Thus, a representation z_i containing global dependency information is obtained. This mechanism can highlight the most critical part for delay prediction based on multi-scale convolutional features.

To further enhance the model's ability to represent features of different dimensions, a feature fusion layer is introduced to combine the convolution output and the attention output. Let the multi-scale convolution result be $H \in R^{T \times d_h}$ and the attention output be $Z \in R^{T \times d_z}$. The fusion is represented as:

$$F = \tanh(W_h H + W_z Z + b_f)$$

Where W_h, W_z is the learnable weight, b_f is the bias vector, and $\tanh(\cdot)$ is the nonlinear function. The fused features not only contain local convolution patterns but also incorporate global dynamic dependency information, providing a richer representation space for subsequent prediction layers.

Finally, the prediction layer maps the fused features to the regression space of the delay value. Specifically, the prediction function is defined as:

$$\hat{y}_t = W_o F_t + b_o$$

Where $\hat{y}_t$ represents the prediction delay at time step t, and W_o and b_o are the output layer parameters. To optimize the performance of the model, the mean square error is used as the objective function:

$$L = \frac{1}{T} \sum_{t=1}^{T} (y_t - \hat{y}_t)^2$$

Where y_t represents the actual latency value, and $\hat{y}_t$ represents the predicted value. This loss function effectively measures the gap between the predicted result and the actual value. It continuously optimizes the model parameters through backpropagation, gradually improving the model's latency prediction capabilities.

4. Performance Evaluation

4.1 Dataset

This study uses the AIOps 2022 Microservice Call Trace Dataset as the sole data source. The dataset comes from a real distributed microservice platform. It collects request-level trace data and host or container-level runtime metrics over several consecutive days. It covers key components such as gateways, business services, and downstream dependencies. Core fields include traceId, spanId, parentSpanId, service, endpoint, startTime, endTime, duration, status, and several key-value tags. It also provides CPU, memory, and network I/O metrics aggregated at fixed time intervals, along with metadata of service dependencies. The dataset has been de-identified and contains no personally identifiable information. It is suitable for academic research and performance analysis.

For the backend API latency prediction task, the dataset directly provides request-level latency signals. The duration of the root span at the gateway or server side can be used as the end-to-end API latency. From the same trace, it is also possible to extract contextual attributes such as cascade depth, critical path length, and error rate. With the synchronized resource monitoring sequences, system load, network fluctuation, and request patterns can be aligned on a unified timeline. This forms a multimodal feature view that combines request-level latency, system state, and call topology. The dataset naturally covers diverse conditions such as peak traffic, periodic fluctuations, and sudden anomalies, which provides sufficient variety for studying multi-scale patterns and the importance of attention weighting.

To ensure reusability and temporal consistency, the data is organized chronologically, preserving original timestamps and service dependency graphs. This makes it convenient to perform aggregation and alignment based on sliding windows. The official documentation specifies the meaning of each field and the record format. The dataset has sufficient scale and diverse business types. It supports fine-grained modeling of single-interface latency as well as aggregated analysis at the service set or business domain level. Overall, the dataset has wide coverage, clear structure, and complete metrics. It matches well with the three key elements of backend API latency modeling: input, context, and target. It provides a stable and reliable basis for further methodological research and systematic comparison.

4.2 Experimental Results

This paper first conducts a comparative experiment, and the experimental results are shown in Table 1.

Table 1. Comparative experimental results

Method	MSE	MAE	R^2	MAPE
CNN-Transformer[8]	0.018	0.091	0.912	6.8%
LSTM-CNN[9]	0.022	0.104	0.887	7.5%
ITransformer[10]	0.016	0.087	0.923	6.2%
TimeMixer[11]	0.015	0.082	0.931	5.9%
Ours	0.012	0.073	0.947	5.1%

From the results of the comparative experiments, it can be seen that traditional hybrid structures, such as LSTM-CNN, show limited performance in latency prediction. Although this method can capture temporal features and local patterns to some extent, its weakness in modeling multidimensional dependencies and global relationships leads to higher MSE and MAE than other methods. At the same time, the R^2 is also lower, only reaching 0.887. This indicates that in complex backend API scenarios, traditional methods cannot fully exploit

potential multi-scale features and nonlinear dynamic relationships.

In contrast, the CNN-Transformer demonstrates a stronger ability in modeling temporal dependencies. With the advantage of self-attention in global feature capture, its R^2 rises to 0.912, and error metrics also decrease. However, this method still shows limitations in extracting local short-term patterns. The MAE remains at the level of 0.091, which suggests that its responsiveness to rapid fluctuations in latency sequences is not ideal. In complex prediction problems with multi-scale and multi-factor interactions, relying only on global modeling remains insufficient.

The emerging iTransformer and TimeMixer methods further improve prediction accuracy. iTransformer performs well in capturing both long-term and short-term dependencies, which significantly reduces MSE and MAE compared with the previous two methods, achieving an R^2 of 0.923. TimeMixer shows even greater advantages in feature interaction and temporal dynamics modeling. Its R^2 increases to 0.931, and MAPE drops to only 5.9 percent. This demonstrates that the method can better adapt to diverse latency patterns in large-scale distributed systems. The trend reveals the importance of flexible feature fusion and dynamic dependency modeling for latency prediction.

On this basis, the proposed method achieves the best overall performance. MSE decreases to 0.012, MAE decreases to 0.073, R^2 rises to 0.947, and MAPE reaches the lowest value of 5.1 percent. This result shows that the combination of multi-scale convolution and attention mechanisms can effectively capture both fine-grained local features and long-term global dependencies in latency sequences. By dynamically focusing on key patterns, it further improves prediction accuracy. This not only verifies the effectiveness of the proposed method but also highlights its application potential in backend API latency prediction. It provides strong support for intelligent operations and system performance assurance.

This paper further presents an experiment on the sensitivity of the weight attenuation coefficient to MAE, and the experimental results are shown in Figure 2.

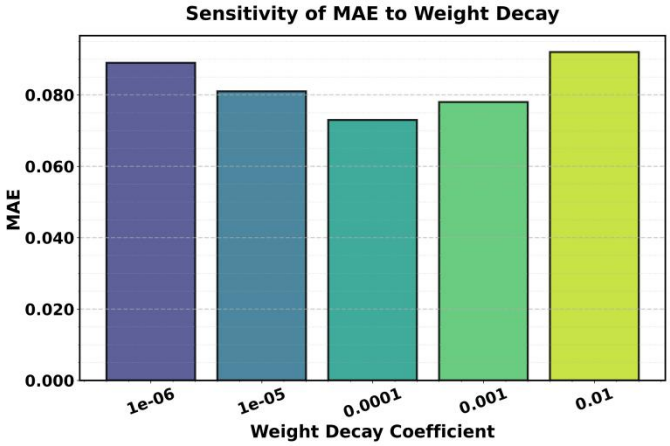

Figure 2. Sensitivity experiment of the weight decay coefficient to MAE

The experimental results show that the weight decay coefficient has a clear sensitivity to MAE performance. When the weight decay is too small, such as 1×10^{-6}, the model suffers from insufficient parameter constraints and tends to overfit. This keeps MAE at a relatively high level and prevents the model from fully capturing regular patterns in latency prediction. With a moderate increase in weight decay, the generalization ability of the model improves.

When the weight decay is set to 1×10^{-4}, MAE reaches the lowest value. This indicates that under this parameter, the model achieves a good balance between preventing overfitting and maintaining sufficient expressive power. It also suggests that the combination of multi-scale convolution and attention mechanisms requires a reasonable regularization coefficient. This avoids excessive model complexity while ensuring that key features can be effectively captured during training. The result confirms the importance of moderate regularization in latency prediction tasks.

When the weight decay further increases to 1×10^{-3} or even 1×10^{-2}, MAE begins to rise again. This shows that overly strong regularization weakens the model's ability to learn complex features in latency data. In this case, the model may fail to represent nonlinear and multi-scale dependencies, which results in poor predictive performance. This trend indicates that excessive parameter constraints can lead to underfitting and reduce prediction accuracy.

Overall, the analysis shows that the weight decay coefficient in latency prediction is not better when larger. Instead, there exists a relatively optimal range. Moderate weight decay can effectively improve the prediction performance of backend API latency. Very small or very large values both reduce generalization ability. Therefore, in practical deployment, weight decay needs to be tuned for different scenarios to ensure stable and accurate prediction in complex distributed system environments.

This paper also presents an experiment on the sensitivity of sampling frequency to R^2, and the experimental results are shown in Figure 3.

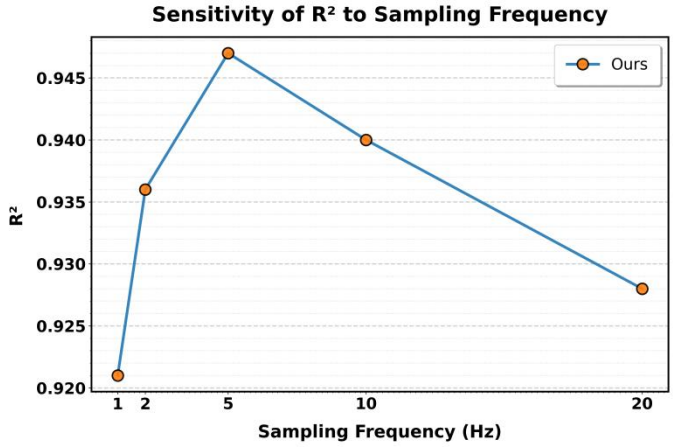

Figure 3. Experiment on the Sensitivity of Sampling Frequency to R^2

The experimental results show that the sampling frequency has a strong sensitivity to the performance of R^2. When the sampling frequency is too low, such as 1 Hz, the model fails to capture the dynamic changes of the latency sequence. This

results in insufficient feature representation, and R^2 remains at the level of 0.921. This indicates that low-frequency sampling, although it reduces data redundancy and storage overhead, significantly weakens the model's ability to represent complex dependencies.

When the sampling frequency increases to 5 Hz, R^2 reaches the highest value of 0.947. This shows that this frequency can preserve the dynamic features of the sequence while avoiding the introduction of excessive noise. As a result, the model achieves balanced modeling of local details and global dependencies. With the joint effect of multi-scale convolution and attention mechanisms, the model can optimally extract and fuse latency features at this sampling frequency.

When the sampling frequency continues to rise to 10 Hz or 20 Hz, R^2 decreases to 0.940 and 0.928, respectively. This indicates that an excessively high sampling frequency introduces large amounts of redundant or even noisy features. The model is disturbed during the learning process, and the generalization ability decreases. This trend shows that simply relying on high-frequency sampling cannot guarantee better predictive performance. On the contrary, it may cause the model to lose focus on key patterns in complex scenarios.

Overall, the choice of sampling frequency is directly related to the performance of latency prediction models. Too low a frequency leads to information loss, while too high a frequency introduces noise. Only within a moderate range can the model maintain high levels of fit and generalization. The experimental results highlight the importance of parameter sensitivity analysis. They also indicate that in practical deployment, sampling frequency should be carefully tuned according to the dynamic characteristics and workload of the system, to ensure accurate and stable latency prediction.

5. Conclusion

This study focuses on the problem of backend API latency prediction and proposes a modeling framework that combines multi-scale convolution with attention mechanisms. By extracting local details and global trends of latency signals at multiple scales and dynamically highlighting key patterns with attention, the model can more comprehensively characterize latency features in complex distributed systems. Experimental results show that the proposed method achieves superior performance across multiple metrics, verifying its effectiveness and practical value. This work not only enriches the theoretical foundation of latency prediction but also provides new solutions for backend performance modeling.

From a broader perspective, this study reveals the importance of multi-scale feature capture and dynamic weight allocation in complex system performance modeling. Traditional methods often struggle with nonlinear dependencies and high-dimensional features. The proposed framework addresses this limitation by combining convolution and attention, effectively overcoming the bottleneck. The results provide new insights for academic research and offer a theoretical basis for intelligent operations in engineering practice, demonstrating wide applicability across different system scenarios.

At the application level, the findings of this study provide practical latency prediction tools for cloud platforms, distributed service architectures, and microservice management. By accurately predicting potential latency fluctuations, systems can take proactive actions such as resource scheduling, load balancing, or fault tolerance to ensure business continuity and user experience. Furthermore, the approach can be extended to other time-series modeling domains, such as network traffic monitoring, industrial equipment anomaly detection, and real-time financial risk assessment, providing strong support for cross-domain intelligent prediction.

Future research can be extended in three directions. First, graph neural networks or spatiotemporal modeling mechanisms can be incorporated to capture more complex service dependencies. Second, adaptive regularization or meta-learning mechanisms can be introduced during training to improve robustness in dynamic environments. Third, cross-system or cross-scenario transfer capabilities can be explored, allowing latency prediction methods to be applied beyond a single platform to diverse distributed environments. Advancing these directions will promote the development of intelligent operations and autonomous performance management and expand the value of latency prediction on a broader scale.

References

[1] Zhou X, Ye J, Zhao S, et al. EffiCANet: Efficient Time Series Forecasting with Convolutional Attention[J]. arXiv preprint arXiv:2411.04669, 2024.

[2] Zhang Y, Yan J. Crossformer: Transformer utilizing cross-dimension dependency for multivariate time series forecasting[C]//International Conference on Learning Representations. 2023.

[3] Jin T. Attention-based temporal convolutional networks and reinforcement learning for supply chain delay prediction and inventory optimization[C]//2024 International Conference on Image Processing, Computer Vision and Machine Learning (ICICML). IEEE, 2024: 1527-1531.

[4] Zou Z, Yan X, Yuan Y, et al. Attention mechanism enhanced LSTM networks for latency prediction in deterministic MEC networks[J]. Intelligent Systems with Applications, 2024, 23: 200425.

[5] Farreras M, Soto P, Camelo M, et al. Improving network delay predictions using gnns[J]. Journal of Network and Systems Management, 2023, 31(4): 65.

[6] Wu H, Xu J, Wang J, et al. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting[C]//Advances in Neural Information Processing Systems. 2021.

[7] Li F, Guo S, Han F, et al. Multi-scale dilated convolution network for long-term time series forecasting[J]. arXiv preprint arXiv:2405.05499, 2024.

[8] Shen L, Wang Y. TCCT: Tightly-coupled convolutional transformer on time series forecasting[J]. Neurocomputing, 2022, 480: 131-145.

[9] Qin Y, Song D, Chen H, et al. A dual-stage attention-based recurrent neural network for time series prediction[C]//Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence. 2017: 2627-2633.

[10] Liu Y, Hu T, Zhang H, et al. itransformer: Inverted transformers are effective for time series forecasting[J]. arXiv preprint arXiv:2310.06625, 2023.

[11] Wang S, Wu H, Shi X, et al. Timemixer: Decomposable multiscale mixing for time series forecasting[J]. arXiv preprint arXiv:2405.14616, 2024.