SD-MoE: Scenario-Driven MoE Forecasting for Intelligent Elastic Scaling in Cloud Clusters

Xianzhao Guo[1], Weipeng Cao[1,*], Minxian Xu[2], Dachuan Li[3], Chuanfei Xu[1], Xi Tao[1], Zhong Ming[1]
[1]*Guangdong Laboratory of Artificial Intelligence and Digital Economy (Shenzhen), Shenzhen, China*
[2]*Shenzhen Institutes of Advanced Technology, Shenzhen, China*
[3]*Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology, Shenzhen, China*
*Corresponding author. Email: caoweipeng123@gmail.com

***Abstract*—Traditional CPU capacity planning in cloud computing often depends on static heuristic rules, which struggle to accommodate temporal dynamics and heterogeneous workload patterns, frequently resulting in resource overprovisioning or performance degradation during demand surges. To address these challenges, we propose SD-MoE, a learning-based framework that integrates scenario-aware sparse expert mechanisms into CPU usage forecasting. The framework comprises four key components: (1) a dual-branch input pipeline that jointly models historical periodicity and real-time telemetry to capture long-term trends and short-term fluctuations; (2) a cross-dimensional transposed embedding module that preserves structured temporal–metric dependencies while mitigating noise from heterogeneous scales; (3) a hierarchical dynamic-routing Mixture of Experts (MoE) that replaces the conventional feed-forward network with four timestamp-activated expert types (weekday-daytime, weekday-nighttime, holiday-daytime, holiday-nighttime), enabling time-specific behavior modeling while retaining global knowledge; and (4) expert grouping and semantic constraints aligned with business scenarios to improve interpretability and facilitate operational integration. Experiments on large-scale production traces demonstrate that SD-MoE significantly enhances peak-load sensitivity and reduces overprovisioning. Notably, it achieves an average R-value of 0.3845, outperforming the strongest baseline by over 13%.**

***Index Terms*—CPU Capacity Forecasting, Scenario-Aware Expert Model, Mixture of Experts, Elastic Scaling**

I. Introduction

Intelligent CPU capacity management aimed at cost reduction and QoS assurance through accurate CPU usage forecasting has become a critical research topic in cloud computing. Cloud workloads are highly dynamic and heterogeneous. Static configurations or simple threshold-based rules often result in either overprovisioning or underprovisioning, leading to inflated operational costs or SLA violations, respectively [2], [3]. Early autoscaling approaches were primarily threshold-based or schedule-based, such as Kubernetes' native Horizontal Pod Autoscaler (HPA) and CronHPA. Although simple to implement, these mechanisms suffer from inherent response latency or limited flexibility when handling bursty traffic and complex periodic patterns. With the advancement of machine learning, cloud elasticity has gradually evolved toward prediction-driven paradigms. Representative systems such as EHPA and AHPA leverage historical data to forecast fine-grained CPU utilization patterns and initiate proactive scaling. However, in real-world production environments, their effectiveness remains constrained by forecasting accuracy and scenario adaptability [29].

Despite the advantages of prediction-driven methods over purely reactive mechanisms, two fundamental challenges remain in practical deployment: forecasting accuracy and multi-scenario adaptability. These challenges are discussed below, followed by the corresponding design considerations.

A. Challenges in Forecast Accuracy and Response Latency

Existing forecasting models exhibit limitations in capturing long-term temporal dependencies and satisfying operational constraints. Traditional time-series models, such as ARIMA, primarily focus on short-term correlations and often fail to model daily or weekly periodic patterns effectively. Transformer-based architectures demonstrate superior long-sequence modeling capability, yet they are typically designed as general-purpose frameworks and may not explicitly incorporate cloud-specific operational constraints, such as cold-start latency and asymmetric error costs [9], [15], [27].

Moreover, most learning objectives employ symmetric loss functions that treat overestimation and underestimation equally. In practical cloud operations, however, underestimation leads to task queuing and SLA violations, incurring significantly higher penalties than moderate overestimation. Therefore, forecasting objectives must explicitly account for asymmetric risk sensitivity. In addition, CPU resource instantiation requires non-negligible cold-start time, which demands appropriate prediction lead time. Improper calibration of this lead time may either waste resources due to excessive anticipation or fail to eliminate latency if anticipation is insufficient.

B. Challenges in Multi-Scenario Adaptation

The heterogeneity and temporal variability of cloud workloads make it difficult for a single dense model to perform well across all scenarios. Forcing diverse workload patterns into a unified representation often leads to feature interference and degraded generalization. The Mixture-of-Experts (MoE) paradigm provides a modular specialization mechanism that enables capacity allocation conditioned on data distributions. However, learnable routing mechanisms may introduce instability, expert load imbalance, and additional engineering

This work was supported by Guangdong Basic and Applied Basic Research Foundation (2025A1515011259).

complexity. Furthermore, if true sparsity is not achieved during inference, computational overhead may approach or even exceed that of an equally parameterized dense model.

To balance modeling flexibility and engineering practicality, we adopt a scenario-driven deterministic allocation strategy. Strong temporal priors (weekday/holiday $\times$ daytime/nighttime) are used to partition experts. Routing is implemented via one-hot temporal encoding combined with argmax selection, ensuring stability and interpretability.

Based on this design philosophy, we propose the Scenario-Driven Mixture of Experts (SD-MoE) model to enhance CPU usage forecasting accuracy and operational robustness in heterogeneous, multi-scenario cloud environments. SD-MoE maps temporal scenarios to dedicated expert groups, each consisting of four temporal experts: weekday daytime, weekday nighttime, holiday daytime, and holiday nighttime. A transposed encoding and normalized embedding preprocessing pipeline is employed to eliminate scale discrepancies across resource dimensions. Operational constraints, including cold-start latency and resource fragmentation, are incorporated into both training and inference through asymmetric objectives to enable a "scale out early, scale in conservatively" strategy. In addition, SD-MoE supports online adaptive learning and multi-scale modeling to handle bursty traffic and scenario drift.

The advantages of SD-MoE include fine-grained scenario modeling capability, stable and interpretable routing mechanisms, and deployment friendliness under operational constraints. Potential limitations lie in the dependence of temporal partitioning on predefined semantic categories and the increased structural complexity compared to minimal baselines, although parameter sharing and sparsification mitigate inference overhead. Experimental results demonstrate that SD-MoE consistently improves forecasting accuracy, resource utilization efficiency, and SLA compliance across diverse cloud workloads, while exhibiting superior robustness under scenario drift. The main contributions of this work are summarized as follows:

- We propose SD-MoE, which allocates dedicated expert groups according to temporal and operational scenarios, enabling fine-grained CPU usage forecasting and adaptive management for heterogeneous cloud workloads.
- We design a four-period expert structure (weekday daytime, weekday nighttime, holiday daytime, holiday nighttime) to capture CPU-specific characteristic behaviors under different temporal scenarios, thereby improving both short-term and long-horizon forecasting performance.
- We introduce a transposed encoding and normalized embedding preprocessing pipeline to construct scale-invariant multi-resource representations, enhancing model transferability and interpretability.
- Beyond conventional metrics (MAE and MSE), we propose the Resource Utilization Efficiency Ratio (R-value) to quantify scaling efficiency under the condition of no underestimation:

$$R = \frac{\text{Area(Actual)}}{\text{Area(Predicted)}} = \frac{\displaystyle\int_{t_0}^{t_1} \text{Actual}(t)\, dt}{\displaystyle\int_{t_0}^{t_1} \text{Predicted}(t)\, dt}$$

  where $\text{Actual}(t)$ denotes actual CPU resource usage, $\text{Predicted}(t)$ denotes the predicted value, and $[t_0, t_1]$ represents the prediction window.

The remainder of this paper is organized as follows: Section II provides a detailed review of related work. Sections III and IV introduce the SD-MoE method and present the corresponding experimental setup and results, respectively. Finally, Section V summarizes the work.

II. Related Work

A. Elastic Scaling of Cluster Capacity

The evolution of elastic scaling progressed from proactive reservation based on high-watermark thresholds to threshold-triggered scaling based on real-time metrics [4]. To address the latency and poor bursty traffic handling of threshold-based methods, prediction-driven approaches were introduced. Early forecasting used time-series models like ARIMA, but these struggled with mixed patterns and lacked cost-penalty mechanisms [26]. Subsequently, machine learning and deep learning (e.g., LSTM) improved prediction accuracy; however, achieving genuine energy efficiency optimization remains difficult without holistically considering renewable energy, workload characteristics, and multi-layer resource interdependencies [21].

However, existing predictive methods are still constrained by data dependency, incomplete decomposition of multi-scale temporal patterns, and a lack of asymmetric cost-awareness, making it difficult to balance task underestimation and resource waste [17], [19], [20]. Hybrid methods combining prediction with convex optimization or reinforcement learning face scalability challenges in large-scale clusters, including high computational overhead, poor real-time responsiveness, and vulnerability to prediction uncertainty [11].

B. Time-Series Forecasting Models

Traditional statistical models (e.g., exponential smoothing and ARIMA) rely on linear assumptions, failing to capture nonlinear dynamics, often leading to systematic overprovisioning during stable periods and underestimation during peak demand [7]. Machine learning approaches (e.g., regression models and SVM), while attempting to improve accuracy through feature engineering, are constrained by manual feature dependency and temporal resolution limits, making it difficult to jointly model long- and short-term dependencies. This results in insufficient lead time for resource activation and suboptimal scaling decisions [1].

Deep learning models (specifically RNNs like LSTM/GRU) demonstrate enhanced capability in capturing nonlinear temporal patterns. Nevertheless, they face gradient vanish-

ing/explosion, which hinders the modeling of long-term dependencies, and sequential processing bottlenecks. This exacerbates the risk of demand underestimation during high traffic, resulting in task queuing and service degradation [10]. esDNN [24] mitigates gradient issues by updating GRU gates, but its effectiveness remains limited under multi-scale and non-stationary conditions [23].

Recent Transformer architectures excel at long-range dependencies but face practical challenges including high computational overhead and inefficient handling of sparse patterns, which limit real-time deployment. Emerging variants (e.g., Informer [27], Autoformer [22], UniTime [13] and iTransformer [15]) attempt to address some of these issues through innovations like decomposition-based modeling and autocorrelation attention. However, critical trade-offs among scalability, interpretability, and accuracy remain unresolved.

C. Mixture-of-Experts (MoE) Models

The MoE architecture has emerged as a pivotal paradigm for scaling model capacity while preserving computational efficiency. Early work introduced sparsely gated MoE layers that activate only a subset of experts, effectively controlling inference cost in ultra-large models and supporting the development of large-scale AI systems. However, these initial designs faced several limitations, including substantial communication overhead, routing imbalance, and training instability, which restricted scalability in distributed environments. A major advancement followed with the introduction of the Switch Transformer, which simplified routing from top-k to top-1 selection. This modification reduced computation and communication costs while achieving improved load balancing [6]. Despite these improvements, issues such as expert load jitter and system-level integration challenges continue to pose difficulties in large-scale deployments. Subsequent research has therefore focused on refining routing strategies and enhancing training stability. One line of work introduced a dynamic routing mechanism in which experts proactively select inputs to achieve more balanced load distribution [28]. Another direction incorporated regularization and capacity-control techniques to promote more stable expert specialization across heterogeneous tasks, ultimately improving model generalization [5]. Although these advancements have substantially improved the efficiency and performance of MoE architectures in language modeling, real-time elastic scaling scenarios introduce additional constraints. In such settings, low-latency inference, lightweight model design, and sensitivity to bursty temporal dynamics become critical. Recent studies have extended MoE architectures to time-series forecasting. For example, linear modeling frameworks augmented with multiple experts and routing mechanisms have demonstrated the ability to partially capture non-stationary patterns [16], though they still struggle to balance complexity and real-time responsiveness in multi-period environments. Another line of work employs sparse MoE structures to build large-scale pre-trained temporal models that capture both short- and long-term dependencies [18]. Nevertheless, due to scheduling delays and engineering complexity, deploying such models in real-time production systems remains challenging. More recent attempts explore token-level expert specialization to improve forecasting accuracy, but simultaneously reveal constraints related to routing jitter, training-to-deployment overhead, and robustness under bursty load conditions [14]. The latest ternary-routing approach further extends traditional binary expert selection by introducing accept, reject, and defer states, achieving improved load balancing and communication efficiency while preserving the benefits of sparse activation [25]. Additionally, the ST-MoE framework [12] utilizes a spatio-temporal gating network as a plug-in to eliminate prediction bias across heterogeneous road segments by tailoring experts to specific individual patterns.

Despite notable advances, the literature exhibits three persistent structural gaps that hinder practical autoscaling in heterogeneous cloud environments in Table I. Rule- and threshold-based elasticity remains attractive for its low deployment cost and fast reaction, yet it depends on manual calibration and offers no principled mechanism to balance cold-start risk and sustained overprovisioning under bursty demand [4]. Forecast-driven schemes improve provisioning timing but depend critically on prediction quality and typically omit asymmetric cost treatment between over- and under-provisioning; classical statistical models fail under abrupt, nonlinear dynamics while feature-driven machine-learning methods and monolithic deep networks struggle to represent short-term bursts and long-range trends simultaneously, incurring substantial overhead and uncertain behavior in low-latency control loops [15], [21], [26]. MoE-based approaches increase representational capacity via sparse activation but introduce routing imbalance, communication and synchronization overhead, inference jitter, and significant engineering complexity that complicate training-to-production transfer and real-time deployment [14], [18], [25]. The combined effect is an unresolved tension among accuracy, latency, and operational constraints, exacerbated by a lack of unified benchmarks (e.g., energy-aware, end-to-end latency, and burst-resilience metrics) for fair comparison [8], [17], [21]. SD-MoE is designed to address these gaps through scenario-driven architectural decomposition, scale-invariant multi-resource encoding, asymmetric loss formulation, and online adaptation—enabling scenario-specific specialization and more robust, low-latency scaling decisions in production settings.

III. Detailed Architecture and Mechanisms of SD-MoE

Our SD-MoE framework is built upon the iTransformer architecture [15], a decoder-only variant of the Transformer. As illustrated in Fig. 1, the design comprises three core components: **(a) Multi-period historical fusion**, where real-time data, timestamps, and historical periodic patterns are integrated through a sliding-window mechanism; **(b) Reverse time encoding**, in which multivariate inputs are encoded as independent tokens to capture temporal dependencies across dimensions; and **(c) Scenario-driven MoE module**, where the conventional feed-forward network (FFN) is replaced with

TABLE I: Comparison of Related Work on Forecasting for Intelligent Elastic Scaling

Category	Representative Methods	Strengths	Limitations
Elastic Scaling	Threshold-based Scaling [4]	Low cost; fast reaction to metric changes; easy to deploy.	Static tuning; cold-start latency; weak under bursty traffic.
	Statistical Prediction [26]	Captures periodicity; improves provisioning timing.	Poor performance under mixed workloads; lacks asymmetric penalties.
	Machine Learning Prediction [21]	Higher forecasting accuracy; enables cost-aware scaling.	High computation; sensitive to prediction errors; limited real-time scalability.
Forecasting Models	Classical Statistical Models (ARIMA, ES [7])	Lightweight; interpretable; effective on stable loads.	Linear assumptions; weak for nonlinear spikes and non-stationarity.
	Machine Learning [1]	Improved feature-driven accuracy.	Heavy reliance on manual features; cannot capture multi-scale dependencies.
	RNN / LSTM [10], [23]	Learns nonlinear temporal patterns; adaptive gating.	Sequential bottlenecks; long-horizon modeling difficulty.
	Transformer Variants (Informer, Autoformer, iTransformer) [15], [22], [27]	Strong long-range modeling; decomposition-based designs.	High computational overhead; deployment challenges.
MoE Architectures	Classical Sparsely-Gated MoE	Scales model capacity efficiently via sparse activation.	Routing imbalance; communication overhead; training instability.
	Switch Transformer [6]	Reduced routing cost; simpler top-1 selection.	Expert load jitter; system-level integration challenges.
	Temporal MoE (Time-MoE, MoLE, large temporal MoE) [14], [16], [18]	Captures partial non-stationarity; supports temporal specialization.	Deployment complexity; limited real-time responsiveness under multi-period workloads.
	Advanced Routing (Ternary routing) [25]	Better load balancing and communication efficiency.	Increased design complexity; robustness under bursty workloads remains open.

an MoE architecture to improve cross-scenario generalization. Under different scenarios, distinct expert groups are activated, each containing four temporal experts, with different timestamps triggering different temporal experts.

A. Multi-period Sliding Historical Fusion Enhanced by Temporal Patterns

This module addresses the dual challenge of short-term responsiveness and long-term periodicity in cloud resource forecasting. Real-time metrics (CPU, memory, and timestamps) are collected through the cluster monitoring system and stored in object storage services for iterative model updates. For example, in CPU utilization forecasting: (1) Input composition: a two-hour historical sequence (9:00 AM–11:00 AM) is combined with real-time data and timestamps. (2) Prediction window: the model predicts the subsequent 40-minute interval (11:00 AM–11:40 AM). (3) Historical periodic aggregation: historical data are aggregated from the same time window over the past m weeks (typically m = 4 or m = 8).

As shown in Fig. 2, the cluster workload exhibits noticeable fluctuations. To enhance the learning of periodic patterns, we adopt a sliding-window strategy that shifts the historical window forward by 40 minutes (9:40 AM–11:40 AM). This design enables the model to: (1) capture multi-scale temporal dependencies (short-term fluctuations and long-term cycles), and (2) learn future-aligned periodic patterns through shifted historical sequences. The mathematical formulation is defined as:

$$P_{out} = F(D_{real-time} \oplus D_{historical}) \tag{1}$$

$D_{real-time}$ and $D_{historical}$ denote the real-time data and historical periodic data, respectively, while P_{out} represents the model used to generate the prediction sequence. The definitions of $D_{real-time}$ and $D_{historical}$ are as follows:

$$D_{real-time} = \{\{C_t, T_t\} | t = 1, \ldots, N\} \tag{2}$$

$$D_{historical} = \sum_{l=1}^{m} \{\{(C_{t-l+p}, T_{t-l+p})\}\} \tag{3}$$

where C and T represent the CPU and timestamp, respectively, and the sequence length is N, where l and p denote the periodic length (e.g. one week) and the prediction length, respectively.

B. Inverted Temporal Encoding and Scale-Invariant Feature Representation

In multivariate time-series forecasting within cloud environments, a key challenge lies in handling heterogeneous metrics, such as CPU and memory utilization, which often exhibit vastly different numerical scales. Standard Transformer models perform tokenization along the temporal dimension, mixing information from multiple metrics within a single token. This leads to scale interference and hinders the model's

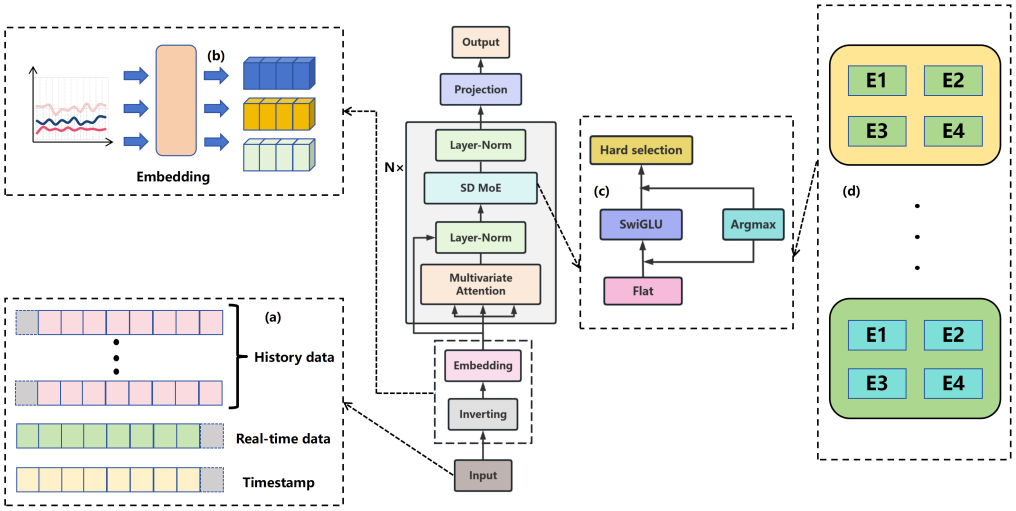

Fig. 1: The network structure of the proposed SD-MoE

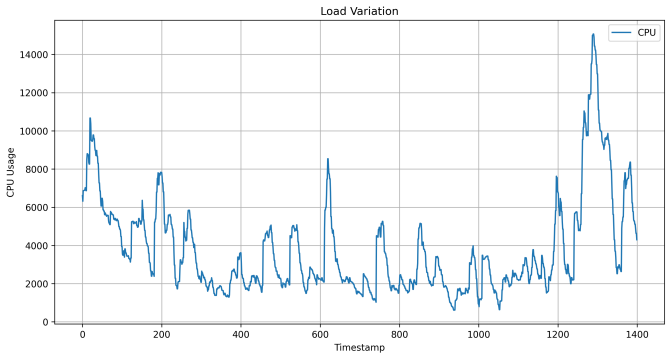

Fig. 2: Load-Variation

ability to capture the complete temporal patterns of each individual dimension.

To address this issue, we introduce an Inverted Temporal Encoding module, one of the core innovations of the SD-MoE framework, inspired by the iTransformer architecture. As illustrated in Fig. 1, this module first receives the unified input tensor W generated by the "Multi-period Historical Fusion" module described in Section III-A. Unlike traditional designs, we do not treat each time step as a token; instead, we transpose the tensor so that each variable (dimension) becomes an independent token.

This procedure is defined in the algorithm as two steps: Invert and Embed. First, the input tensor W is transformed via a transpose operation:

$$W' = \text{Invert}(W) = W^T \tag{4}$$

where $W \in \mathbb{R}^{L\times M}$ denotes the input tensor formed by concatenating real-time data, timestamps, and historical data, where L represents the sequence length and M denotes the number of variables (metrics). The inversion operation Invert$(*)$, i.e., the transpose T, produces $W' \in \mathbb{R}^{L\times M}$. This operation restructures the data into M tokens, with each token containing the complete sequence information across L time steps. Subsequently, these M variable tokens are fed into the embedding layer.

$$E = \text{Embedding}(W') \tag{5}$$

where Embedding() is a learnable embedding layer that projects each one-dimensional variable sequence $w'_i \in \mathbb{R}^L$ (i.e., the i-th row of W') of length L into a D-dimensional hidden space. This process generates the final token representation $E \in \mathbb{R}^{M\times D}$, which will serve as the input for the subsequent Multivariate Attention module.

C. Scenario-Driven Mixture-of-Experts (SD-MoE) Architecture

To precisely capture the inherent multi-modal temporal patterns in cloud resource utilization, such as the significant differences between weekdays and holidays, or day and night, we replace the standard Feed-Forward Network (FFN) in the encoder layer with a Semantic-Driven Mixture-of-Experts (SD-MoE) module. Unlike architectures that rely on Sparsely-Gated MoE, the core of SD-MoE is a deterministic routing mechanism based on prior domain knowledge. In multivariate time series forecasting, particularly in cloud resource scheduling scenarios, data often exhibits highly multi-modal characteristics. For instance, the resource consumption patterns during weekday daytime are statistically distinct from those during holiday nighttimes. Notably, as shown in Fig 1 (d), we introduce a scenario-driven expert grouping mechanism here: experts are grouped according to the distinct temporal patterns observed in the operational workload (e.g., weekday daytime, weekday nighttime, holiday daytime, and holiday nighttime). Each group typically consists of four experts:

four dedicated specialists committed to modeling the unique dynamics of each period. By aligning the model architecture with actual operational semantics, this design enhances the interpretability and adaptability to different workload regimes, while supporting structured knowledge sharing both within and between groups. This mechanism first parses the input temporal metadata into four semantically explicit channels through a temporal encoding function: weekday daytime, weekday nighttime, holiday daytime, and holiday nighttime. This process generates a semantic routing tensor $T \in \mathbb{R}^{B\times L\times 4}$ (where B is the batch size and L is the sequence length). This tensor acts as a hard instruction for subsequent routing, rather than a trainable weight. In the forward pass of the SD-MoE (ExpandedMoE) module, the inputs X and T are flattened. Subsequently, instead of using Softmax or Top-K, we apply an $\arg\max$ operation to the flattened form of T to determine a unique, deterministic expert index idx_n for each token n:

$$idx_n = \arg\max_{i\in\{0,1,2,3\}} (T_n, i)$$

Concurrently, all four experts E_l (implemented as SwiGLU activation networks) process all input tokens X_{flat} in parallel, yielding four sets of candidate outputs H_l:

$$H_l = E_l(X_{flat})$$

Finally, the module's output Y is directly extracted from H via a Hard Selection mechanism (i.e., advanced indexing) based on idx_n:

$$y_n = H[n, idx_n :]$$

The overall training procedure is summarized in **Algorithm 1**. Given real-time observations $D_{\text{real-time}} \in \mathbb{R}^{B\times L_{rt}\times M}$ and historical observations $D_{\text{historical}} \in \mathbb{R}^{B\times L_{hist}\times M}$, the model first concatenates them along the temporal dimension to obtain a unified tensor $W \in \mathbb{R}^{B\times L\times M}$, where $L = L_{rt} + L_{hist}$. An invert (transpose) operation is then applied to reorganize the variable dimension as the token dimension, yielding $W' \in \mathbb{R}^{B\times M\times L}$. This inverted embedding strategy treats each variable as an independent token while preserving its temporal evolution. After an embedding transformation, the initial hidden representation $E \in \mathbb{R}^{B\times M\times D}$ is obtained.

To enable temporally aware expert selection, the timestamp sequence $T \in \mathbb{Z}^{B\times L_{rt}}$ is first encoded into a one-hot representation $T_{\text{onehot}} \in \mathbb{R}^{B\times L_{rt}\times K}$, where K denotes the number of experts. Since the model operates in the inverted embedding space where tokens correspond to variables, the temporal routing signal is aggregated along the temporal dimension to produce $T_{\text{agg}} \in \mathbb{R}^{B\times K}$. The aggregated routing vector is then broadcast to the variable dimension, resulting in $T_{\text{route}} \in \mathbb{R}^{B\times M\times K}$, which provides structured temporal context for expert routing.

The encoder consists of N stacked blocks. Within each block, the representation E is first processed by a self-attention module to model inter-variable dependencies, followed by residual connection and normalization. Subsequently, a deterministic routing mechanism computes a routing index $idx = \arg\max(T_{\text{route}}, -1)$ for each token. Under this hard routing strategy, each token is assigned to exactly one expert. The selected expert network transforms the token representation, and the outputs are merged to update E. After passing through all encoder blocks, the final representation is projected through a linear layer to produce the predicted future resource usage sequence $U \in \mathbb{R}^{B\times O_{len}\times M_{out}}$.

In terms of computational complexity, let L denote the token length, D the embedding dimension, K the number of experts, and N the number of encoder blocks. For each block, the dominant cost arises from the self-attention module and the expert feed-forward transformation. The self-attention mechanism incurs a time complexity of $\mathcal{O}(L^2D)$. Due to deterministic sparse routing, each token is processed by exactly one expert, and therefore the total expert computation remains $\mathcal{O}(LD^2)$, which is comparable to a standard dense feed-forward network. Consequently, the overall time complexity of the N-block encoder can be expressed as

$$\mathcal{O}_{\text{total}} = N \cdot \left(\mathcal{O}(L^2D) + \mathcal{O}(LD^2)\right).$$

Algorithm 1: SD-MoE Training Procedure

Input : Real-time data $D_{rt} \in \mathbb{R}^{B\times L_{rt}\times M}$;
Historical data $D_{hist} \in \mathbb{R}^{B\times L_{hist}\times M}$;
Timestamp sequence $T \in \mathbb{Z}^{B\times L_{rt}}$;
Prediction length O_{len}

Output: $U \in \mathbb{R}^{B\times O_{len}\times M_{out}}$

for *each batch* **do**
- $W \leftarrow \text{concat}(D_{rt}, D_{hist})$; ; // $W \in \mathbb{R}^{B\times L\times M}$
- $W' \leftarrow \text{Invert}(W)$; ; // $W' \in \mathbb{R}^{B\times M\times L}$
- $E \leftarrow \text{Embedding}(W')$; ; // $E \in \mathbb{R}^{B\times M\times D}$
- $T_{onehot} \leftarrow \text{TimeOneHot}(T)$; ; // $\in \mathbb{R}^{B\times L_{rt}\times E}$
- $T_{agg} \leftarrow \text{Aggregate}(T_{onehot})$; ; // $\in \mathbb{R}^{B\times E}$
- $T_{route} \leftarrow \text{Broadcast}(T_{agg}, M)$; ; // $\in \mathbb{R}^{B\times M\times E}$
- **for** $b = 1$ **to** N **do**
  - $E \leftarrow \text{LayerNorm}(E + \text{SelfAttention}_b(E))$
  - $idx \leftarrow \arg\max(T_{route}, -1)$
  - $E \leftarrow \text{MoE}_b(E, idx)$; // each token processed by one selected expert
- **end**
- $U \leftarrow \text{Projection}(E)$

end

return U

IV. Experiments and Results

A. Datasets

To validate the generalization ability and specialized efficiency of the proposed algorithm, this study employs two real-world, cluster-level datasets sourced from the production environments of leading global cloud service providers. The first, a Huawei Cloud dataset, focuses on traditional cluster-level capacity forecasting, while the second, an Alibaba Cloud

dataset, addresses resource bottlenecks encountered in emerging large-scale Generative AI (GenAI) inference services.

1) Huawei Cloud Cluster Capacity Forecasting Benchmark Dataset: We utilize a production environment dataset exclusively from a leading global cloud service provider, marking the first public release of real cluster-level resource usage data for cloud capacity forecasting research.

This dataset comprises 72 heterogeneous clusters characterized by distinct workload profiles (e.g., batch processing, real-time services) and resource scales. We select CPU utilization forecasting as the primary evaluation metric, as it commonly represents the critical bottleneck in cloud resource allocation.

The data collection pipeline adheres to a strict procedure:

Cluster Monitoring System: An independent monitoring agent is deployed in each cluster to report real-time resource metrics (CPU, Memory, Timestamp) to a message queue at 30-second intervals.

Data Aggregation and Preprocessing: Time series data is grouped by cluster ID and timestamp. To ensure temporal continuity and data integrity, linear interpolation is used to handle missing values.

Input Requirement: To capture sufficient historical periodic patterns, model training mandates at least 4 weeks of historical data. Consequently, four months of raw data were collected for each cluster, allowing for a three-month training period and a one-month validation period.

2) Alibaba Cloud GenAI Service High-Performance Prediction Benchmark Dataset: To further validate the model's efficacy on modern, highly heterogeneous GPU workloads, we introduce the GenAI Serving Top-Down Dataset 2026 (GenTD26). This dataset provides a comprehensive top-down view of large-scale GenAI service systems, capturing performance data across three critical architectural tiers: the application layer, the middleware layer, and the infrastructure layer.

We focus on the infrastructure layer metric within this dataset: GPU memory usage, specifically using the data source pod_gpu_memory_used_bytes_anon.csv. This file records the GPU memory usage for each Kubernetes Pod. Key fields include:timestamp_anon (timestamp), value (GPU memory usage in bytes), and container_ip (MD5 hash of the container IP).

The data organization and experimental setup process are as follows:

Heterogeneous Instance Grouping: Data is aggregated by the container_ip field, treating each IP as an independent inference instance with heterogeneous workload characteristics. Thus, each IP corresponds to a unique CSV time series file.

Data Filtering and Merging: To satisfy the model's requirement for long historical inputs, we only filter out IP files whose sequence length exceeds 1440 time points.

Model Training: All IP files meeting the length requirement are merged into a single large-scale training set, used to train the general and expert parameters of the SD-MoE model.

Model Inference and Generalization Testing: In the inference phase, we randomly select 40 different IP files from the dataset for prediction, rigorously evaluating the model's generalization capability on unseen, highly specialized GenAI inference tasks.

B. Experimental Setup

Cluster Capacity Forecasting Experiment

The experimental design addresses three critical challenges:

(1) Handling Missing Historical Data: 1) Training phase: Samples with missing historical data are dropped to ensure data quality. 2) Inference phase: Missing historical values are imputed using the real-time data input to maintain temporal continuity.

(2) Resource Fragmentation Mitigation: The predicted values are deliberately overestimated to accommodate resource activation latency. This is achieved through an asymmetric penalty mechanism featuring a large underestimation penalty and a smaller overestimation penalty.

(3) Peak Detection Optimization: For clusters with bursty workloads, peaks are extracted every 5 minutes from a 40-minute prediction window to enhance scheduling responsiveness.

AI Inference Service Infrastructure Resource Prediction

For the GPU memory usage forecasting task on the GenAI Serving Top-Down Dataset 2026 (GenTD26), we designed a specialized set of experimental settings to ensure that the prediction results effectively meet the demands of cloud resource scheduling.

(1) Dataset Preparation and Filtering:

Training Phase: To focus on workloads with sufficient historical information and complete periodicity, the data is first grouped by the container_ip field. Only container IP files with a total sequence length greater than 1440 time steps are filtered out. All passing heterogeneous IP sequences are merged into a single large-scale training set.

Inference Phase: 40 randomly selected IP sequences are used for inference validation to assess the model's generalization capability.

(2) Label Design: Segmented Maximum Prediction Traditional point forecasting struggles to cope with the sharp, bursty peaks in GPU memory usage caused by batch processing and model switching. To make the prediction more meaningful for scheduling, the task is transformed into predicting the resource peak value within a future time window:

Segmentation: The total prediction window length is 40 time steps. This window is divided into 10 sub-segments, with each sub-segment containing 4 time steps.

Label Generation: The Target Label for each sub-segment is set as the maximum sequence value of GPU memory usage within that sub-segment.

(3) Loss Function and Fault Tolerance Mechanism: To counter the high risk associated with "underestimation" in resource forecasting (which can lead to service interruption), Quantile Loss (QL) is adopted as the primary optimization objective during the training phase.

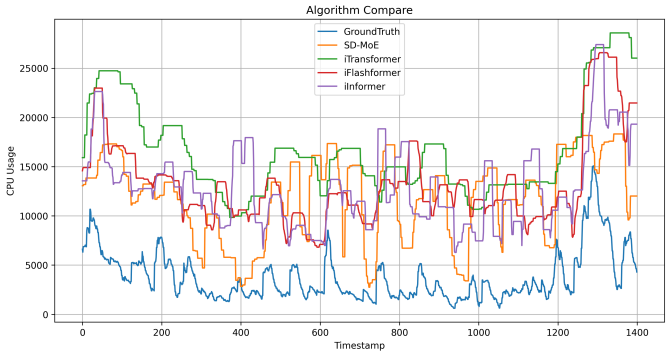

Fig. 3: Example of One-Day Prediction Results for All Algorithms. A comparison of the actual resource usage (blue curve), iTransformer prediction (green curve), iFlashformer (red curve), iInformer (purple curve), and our SD-MoE prediction (orange curve).

The definition of Quantile Loss is as follows:

$$\mathcal{L}_\tau(y,u) = \begin{cases} \tau \cdot (y-u), & \text{if } y \geq u \\ (1-\tau)\cdot(u-y), & \text{if } y < u \end{cases} \tag{6}$$

where y is the true label (i.e., the maximum value of the sub-segment), u is the model's predicted output, and τ is the quantile coefficient. We set the quantile coefficient τ to 0.7. When $\tau > 0.5$, this loss function assigns a higher penalty weight to cases where the true value is greater than the predicted value (i.e., model underestimation). This asymmetric penalization mechanism effectively encourages the model to predict towards a higher quantile boundary, thereby avoiding underestimation and ensuring the prediction results meet the safety margin requirements for GPU memory scheduling.

All experiments were conducted on the Artificial Intelligence Platform of a cloud service provider, and all models used the same hardware configuration and hyperparameters.

C. Main Results

First, we evaluate the overall performance of the proposed SD-MoE algorithm. To ensure the representativeness and robustness of the findings, we randomly selected a set of real-world clusters with varying workload characteristics and resource scales. For this cluster, SD-MoE randomly forecasts resource usage for three consecutive days, effectively simulating a practical scheduling scenario. We conducted a comparative study of our model against several popular Transformer-based time series forecasting algorithms (iTransformer, iFlashformer, and iInformer) for the cluster resource prediction task. All methods were evaluated under the same experimental setup, using the same dataset, environment, and evaluation metrics. Furthermore, the same transposition was applied to the Embedding layer of all models to ensure a fair comparison. As shown in Table II, our model achieved the highest R-value.

As indicated by the orange line in Fig 3, our algorithm can effectively reduce resource waste compared to other algorithms while guaranteeing against underestimation.

TABLE II: Performance Comparison of Different Algorithms on the Cluster Capacity Forecasting Dataset.

Algorithm	**R-value**			**Average**
	Test 1	Test 2	Test 3	
SD-MoE (Ours)	**0.317712**	**0.407932**	**0.427989**	**0.384544**
iTransformer(2024) [15]	0.226773	0.307959	0.295196	0.276643
iFlashformer(2022) [27]	0.282623	0.381990	0.352675	0.339096
iInformer(2021) [9]	0.290981	0.357192	0.344085	0.330753

To rigorously assess the performance boundaries and domain specialization of our SD-MoE framework, we performed a Zero-Shot comparison against Time-MoE, a state-of-the-art pretrained foundation model based on the sparse MoE architecture. Time-MoE was pretrained on a massive dataset of 300 billion timestamps and has a parameter scale of up to 240 million, representing the current high standard for generalization capability in the time series domain. We strictly evaluated Time-MoE in a Zero-Shot setting, meaning its parameters were not updated; it was applied directly to the cluster capacity forecasting dataset. As shown in Table III, leveraging its vast pretrained knowledge, SD-MoE achieved a slight advantage in the raw R-value.

TABLE III: Performance Comparison of SD-MoE and Time-MoE on the Cluster Capacity Forecasting Dataset.

Algorithm	**R-value**			**Average**
	Test 1	Test 2	Test 3	
SD-MoE(Ours)	**0.317712**	**0.407932**	**0.427989**	**0.384544**
Time-MoE(2024) [18]	0.326773	0.329926	0.412981	0.35656

To evaluate the cross-scenario performance of SD-MoE, we randomly selected 40 container_ips from the GenAI Service High-Performance Forecasting Benchmark Dataset for inference. We then calculated two metrics for each algorithm: the proportion of predictions that satisfy the no-underestimation condition and the R-value under this no-underestimation condition. All methods were evaluated under the same experimental setup, utilizing the same dataset, environment, and evaluation metrics. As shown in Table IV, our model achieves the highest proportion of no-underestimation and its R-value approaches 1 in the no-underestimation prediction scenario.

TABLE IV: Performance Comparison of Algorithms.

	Non-Underestimation(%)	**Average R-value**
SD-MoE(Ours)	**90**	**0.9749**
iTransformer(2024) [15]	82	0.9107
iFlashformer(2022) [27]	75	0.9449
iInformer(2021) [9]	71	0.9249

Beyond forecasting accuracy and robustness, inference throughput serves as a pivotal metric for evaluating the feasibility of cluster scheduling algorithms in production environments. In high-concurrency workload scenarios, the

throughput—defined as the number of samples processed per second—directly dictates the system's responsiveness and scalability.

To this end, we benchmarked the inference throughput of SD-MoE against the iTransformer baseline. Extensive inference tests were conducted across 40 representative real-world clusters. As summarized in Table V, while iTransformer theoretically possesses a slight advantage due to its streamlined static FFN architecture, experimental results reveal a negligible gap in processing efficiency. Our SD-MoE maintains a processing speed remarkably close to that of the baseline, with a throughput degradation of only approximately 1.03%.

TABLE V: Detailed Comparison of Inference Throughput (Samples/s) Across 40 Clusters.

Sample	**SD-MoE (Ours)**	**iTransformer(2024) [15]**
Cluster 1	8506.492	8846.752
Cluster 2	8495.299	8665.205
⋮	⋮	⋮
Cluster 40	8418.995	8518.339
Average	**8456.491**	**8544.501**

This high-performance efficiency is primarily attributed to our proposed Deterministic Routing mechanism. Unlike conventional dynamic MoE frameworks, SD-MoE bypasses the execution of complex gating branches and frequent data reshuffling by directly mapping expert indices via timestamps. This allows the model to leverage the high-precision gains of the Mixture-of-Experts architecture with minimal sacrifice in throughput. These results demonstrate that SD-MoE is well-equipped to meet the stringent demands for high-concurrency and real-time processing in large-scale production resource management systems.

To evaluate the generalization ability of SD-MoE across heterogeneous workloads, we conducted an ablation study comparing two MoE versions: the SD-MoE version, which utilizes a deterministic routing mechanism based on prior domain knowledge, and a baseline version, Native-MoE, which employs a traditional Top-2 routing MoE module with ten experts. The evaluation focuses on two key metrics: (1) the percentage of time the prediction curve envelopes the true curve (i.e., the no-underestimation rate), and (2) the R-value (which reflects the impact of resource wastage caused by overestimation).

As shown in Fig 4, when inference is performed under the same dataset, environment, and evaluation metrics, and with the same transposition applied to the Embedding layer, SD-MoE significantly reduces resource wastage compared to Native-MoE while ensuring no resource underestimation occurs. Table VI demonstrates that our model, utilizing a deterministic routing mechanism, can reduce resource wastage and yield greater benefits compared to the traditional MoE architecture.

TABLE VI: Ablation Study: Performance Comparison of SD-MoE and Native-MoE.

Algorithm	**R-value**		
	Test 1	Test 2	Test 3
SD-MoE (Ours)	**0.317712**	**0.407932**	**0.427989**
Native-MoE	0.186474	0.287952	0.251894

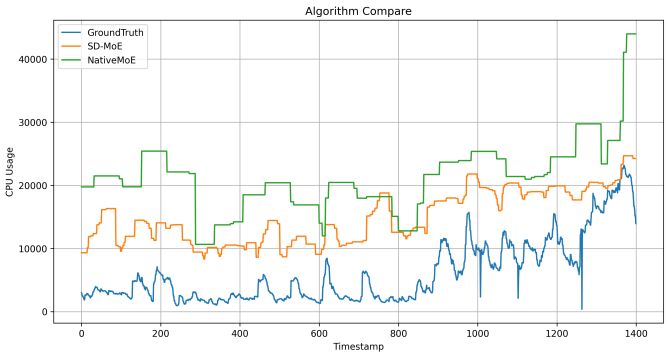

Fig. 4: Example of One-Day Prediction Results for Two Algorithms. A comparison of the actual resource usage (blue curve), the Native-MoE prediction (green curve), and our SD-MoE prediction (orange curve).

V. Conclusions

This work introduces the SD-MoE framework, a novel learning paradigm built on the iTransformer architecture to manage complex and heterogeneous dynamic cloud workloads. The framework achieves a robust balance in temporal modeling through three core components: Multi-period Historical Fusion, which integrates real-time data and long-term periodic patterns for enhanced stability; Inverted Temporal Encoding, which captures cross-dimensional dependencies to mitigate scale interference in multivariate inputs; and a Scenario-Driven MoE module, which replaces standard Feed-Forward Networks by utilizing a rigorous semantic routing mechanism with specialized time-period experts to significantly boost cross-scenario generalization.Empirical evaluation across production traces demonstrates SD-MoE's significant superiority over state-of-the-art time-series models. On cluster capacity prediction benchmarks, SD-MoE achieved over 40% average relative improvement in the Resource Utilization Efficiency Ratio (R-value) compared to its baseline. Furthermore, it exhibited a high safety-efficiency trade-off on GenAI benchmarks, reaching 90% non-underestimation and an average $R = 0.9749$ in safe scenarios, closely approaching ideal efficiency ($R = 1.0$). Ablation studies confirmed the critical role of the semantic routing mechanism, which yielded a 42.5% R-value uplift over the Native-MoE baseline. This advantage underscores the efficacy of domain-specific design, as further validated by a Zero-Shot comparison against the ultra-scale pre-trained Time-MoE foundation model, where SD-MoE secured a slight advantage in the raw R-value.

Furthermore, SD-MoE maintains exceptional computational efficiency, achieving millisecond-level inference latency per sample, which satisfies the stringent real-time requirements of large-scale cloud scheduling environments. From an implementation perspective, the framework's broad coverage across multiple scenarios results in a significantly larger parameter scale compared to traditional dense models, which imposes substantial demands on hardware resources. Nevertheless, because the SD-MoE architecture selectively activates specific experts via timestamps, it successfully maintains inference latency within strict engineering requirements.

As a path forward, future work will focus on integrating richer domain-specific contextual signals to improve routing precision, designing dynamic expert routing mechanisms to adapt to real-time workload phase changes, and developing a hybrid prediction paradigm that combines data-driven SD-MoE with rule-based anomaly detection for enhanced resilience in extreme demand events.

REFERENCES

[1] Doaa Bliedy, Mohamed H Khafagy, and Rasha M Badry. Resource utilization prediction model for cloud datacentre: Survey. *International Journal of Advanced Computer Science & Applications*, 16(3), 2025.

[2] Weipeng Cao, Jiongjiong Gu, Zhong Ming, Zhiyuan Cai, Yuzhao Wang, Changping Ji, Zhijiao Xiao, Yuhong Feng, Ye Liu, and Liang Jie Zhang. Flexible computing: A new framework for improving resource allocation and scheduling in elastic computing. *IEEE Transactions on Services Computing*, 18(1):198–211, 2025.

[3] Weipeng Cao, Xi Tao, Yinghui Pan, Ye Liu, and Zhong Ming. Qos perception for cloud databases: Necessity, trends, and challenges. In *2024 IEEE/ACM 32nd International Symposium on Quality of Service (IWQoS)*, pages 1–2. IEEE, 2024.

[4] Marta Catillo, Umberto Villano, and Massimiliano Rak. A survey on auto-scaling: how to exploit cloud elasticity. *International Journal of Grid and Utility Computing*, 14(1):37–50, 2023.

[5] Damai Dai, Li Dong, Shuming Ma, Bo Zheng, Zhifang Sui, Baobao Chang, and Furu Wei. Stablemoe: Stable routing strategy for mixture of experts. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7085–7095, 2022.

[6] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[7] Binbin Feng and Zhijun Ding. Application-oriented cloud workload prediction: A survey and new perspectives. *Tsinghua Science and Technology*, 30(1):34–54, 2024.

[8] Pasan Bhanu Guruge and YHPP Priyadarshana. Time series forecasting-based kubernetes autoscaling using facebook prophet and long short-term memory. *Frontiers in Computer Science*, 7:1509165, 2025.

[9] Weizhe Hua, Zihang Dai, Hanxiao Liu, and Quoc Le. Transformer quality in linear time. In *International Conference on Machine Learning*, pages 9099–9117. PMLR, 2022.

[10] Yaxuan Kong, Zepu Wang, Yuqi Nie, Tian Zhou, Stefan Zohren, Yuxuan Liang, Peng Sun, and Qingsong Wen. Unlocking the power of lstm for long term time series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 11968–11976, 2025.

[11] Kamlesh Lakhwani, Gajanand Sharma, Ramandeep Sandhu, Naresh Kumar Nagwani, Sandeep Bhargava, Varsha Arya, and Ammar Almomani. Adaptive and convex optimization-inspired workflow scheduling for cloud environment. *International Journal of Cloud Applications and Computing (IJCAC)*, 13(1):1–25, 2023.

[12] Shuhao Li, Yue Cui, Yan Zhao, Weidong Yang, Ruiyuan Zhang, and Xiaofang Zhou. St-moe: Spatio-temporal mixture-of-experts for debiasing in traffic prediction. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 1208–1217, 2023.

[13] Xu Liu, Junfeng Hu, Yuan Li, Shizhe Diao, Yuxuan Liang, Bryan Hooi, and Roger Zimmermann. Unitime: A language-empowered unified model for cross-domain time series forecasting. In *Proceedings of the ACM Web Conference 2024*, pages 4095–4106, 2024.

[14] Xu Liu, Juncheng Liu, Gerald Woo, Taha Aksu, Yuxuan Liang, Roger Zimmermann, Chenghao Liu, Junnan Li, Silvio Savarese, Caiming Xiong, et al. Moirai-moe: Empowering time series foundation models with sparse mixture of experts. In *International Conference on Machine Learning*, pages 38940–38962. PMLR, 2025.

[15] Yong Liu, Tengge Hu, Haoran Zhang, Haixu Wu, Shiyu Wang, Lintao Ma, and Mingsheng Long. itransformer: Inverted transformers are effective for time series forecasting. In *The Twelfth International Conference on Learning Representations*, 2023.

[16] Ronghao Ni, Zinan Lin, Shuaiqi Wang, and Giulia Fanti. Mixture-of-linear-experts for long-term time series forecasting. In *International Conference on Artificial Intelligence and Statistics*, pages 4672–4680. PMLR, 2024.

[17] Zhicheng Pan, Yihang Wang, Yingying Zhang, Sean Bin Yang, Yunyao Cheng, Peng Chen, Chenjuan Guo, Qingsong Wen, Xiduo Tian, Yunliang Dou, et al. Magicscaler: Uncertainty-aware, predictive autoscaling. *Proceedings of the VLDB Endowment*, 16(12):3808–3821, 2023.

[18] Xiaoming Shi, Shiyu Wang, Yuqi Nie, Dianqi Li, Ye Zhou, Qingsong Wen, and Ming Jin. Time-MoE: Billion-scale time series foundation models with mixture of experts. In *Proceedings of the Thirteenth International Conference on Learning Representations*, 2025.

[19] Linfeng Wen, Minxian Xu, Sukhpal Singh Gill, Muhammad Hilman, Satish Narayana Srirama, Kejiang Ye, and Chengzhong Xu. Statuscale: Status-aware and elastic scaling strategy for microservice applications. *ACM Transactions on Autonomous and Adaptive Systems*, 20(1):1–25, 2025.

[20] Linfeng Wen, Minxian Xu, Adel N Toosi, and Kejiang Ye. Temposcale: A cloud workloads prediction approach integrating short-term and long-term information. In *2024 IEEE 17th International Conference on Cloud Computing (CLOUD)*, pages 183–193. IEEE, 2024.

[21] Qingsong Wen, Kai He, Liang Sun, Yingying Zhang, Min Ke, and Huan Xu. Robustperiod: Robust time-frequency mining for multiple periodicity detection. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2328–2337, 2021.

[22] Haixu Wu, Jiehui Xu, Jianmin Wang, and Mingsheng Long. Autoformer: Decomposition transformers with auto-correlation for long-term series forecasting. *Advances in Neural Information Processing Systems*, 34:22419–22430, 2021.

[23] Yuhan Wu, Xiyu Meng, Junru Zhang, Yang He, Joseph A Romo, Yabo Dong, and Dongming Lu. Effective lstms with seasonal-trend decomposition and adaptive learning and niching-based backtracking search algorithm for time series forecasting. *Expert Systems with Applications*, 236:121202, 2024.

[24] Minxian Xu, Chenghao Song, Huaming Wu, Sukhpal Singh Gill, Kejiang Ye, and Chengzhong Xu. esdnn: deep neural network based multivariate workload prediction in cloud computing environments. *ACM Transactions on Internet Technology (TOIT)*, 22(3):1–24, 2022.

[25] Shen Yan, Xingyan Bin, Sijun Zhang, Yisen Wang, and Zhouchen Lin. Tc-moe: Augmenting mixture of experts with ternary expert choice. In *The Thirteenth International Conference on Learning Representations*, 2025.

[26] Haibin Yuan and Shengchen Liao. A time series-based approach to elastic kubernetes scaling. *Electronics*, 13(2):285, 2024.

[27] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 11106–11115, 2021.

[28] Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.

[29] Zhiqiang Zhou, Chaoli Zhang, Lingna Ma, Jing Gu, Huajie Qian, Qingsong Wen, Liang Sun, Peng Li, and Zhimin Tang. Ahpa: adaptive horizontal pod autoscaling systems on alibaba cloud container service for kubernetes. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 15621–15629, 2023.