# MagicScaler: Uncertainty-aware, Predictive Autoscaling

Zhicheng Pan
East China Normal University
Alibaba Group
zcpan@stu.ecnu.edu.cn

Yihang Wang
East China Normal University
Alibaba Group
yhwang@stu.ecnu.edu.cn

Yingying Zhang*
Alibaba Group
congrong.zyy@alibaba-inc.com

Sean Bin Yang
Aalborg University
seany@cs.aau.dk

Yunyao Cheng
Aalborg University
yunyaoc@cs.aau.dk

Peng Chen
East China Normal University
pchen@stu.ecnu.edu.cn

Chenjuan Guo
East China Normal University
cjguo@dase.ecnu.edu.cn

Qingsong Wen
Alibaba Group
qingsong.wen@alibaba-inc.com

Xiduo Tian
Alibaba Group
xiduo.txd@alibaba-inc.com

Yunliang Dou
Alibaba Group
yunliang.dyl@alibaba-inc.com

Zhiqiang Zhou
Alibaba Group
zhouzhiqiang.zzq@alibaba-inc.com

Chengcheng Yang
East China Normal University
ccyang@dase.ecnu.edu.cn

Aoying Zhou
East China Normal University
ayzhou@dase.ecnu.edu.cn

Bin Yang*
East China Normal University
byang@dase.ecnu.edu.cn

## ABSTRACT

Predictive autoscaling is a key enabler for optimizing cloud resource allocation in Alibaba Cloud's computing platforms, which dynamically adjust the *Elastic Compute Service* (ECS) instances based on predicted user demands to ensure Quality of Service (QoS). However, user demands in the cloud are often highly complex, with high *uncertainty* and *scale-sensitive* temporal dependencies, thus posing great challenges for accurate prediction of future demands. These in turn make autoscaling challenging—autoscaling needs to properly account for demand uncertainty while maintaining a reasonable trade-off between two contradictory factors, i.e., low instance running costs vs. low QoS violation risks.

To address the above challenges, we propose a novel predictive autoscaling framework *MagicScaler*, consisting of a Multi-scale attentive Gaussian process based predictor and an uncertainty-aware scaler. First, the predictor carefully bridges the best of two successful prediction methodologies—multi-scale attention mechanisms, which are good at capturing complex, multi-scale features, and stochastic process regression, which can quantify prediction uncertainty, thus achieving accurate demand prediction with quantified uncertainty. Second, the scaler takes the quantified future demand uncertainty into a judiciously designed loss function with stochastic constraints, enabling flexible trade-off between running costs and QoS violation risks. Extensive experiments on three clusters of Alibaba Cloud in different Chinese cities demonstrate the effectiveness and efficiency of *MagicScaler*, which outperforms other commonly adopted scalers, thus justifying our design choices.

**PVLDB Reference Format:**
Zhicheng Pan, Yihang Wang, Yingying Zhang, Sean Bin Yang, Yunyao Cheng, Peng Chen, Chenjuan Guo, Qingsong Wen, Xiduo Tian, Yunliang Dou, Zhiqiang Zhou, Chengcheng Yang, Aoying Zhou, Bin Yang. MagicScaler: Uncertainty-aware, Predictive Autoscaling. PVLDB, 16(12): 3808 - 3821, 2023.
doi:10.14778/3611540.3611566

*Corresponding authors.

## 1 INTRODUCTION

Cloud computing [26, 28, 34, 37] is an emerging computing paradigm that provides a plethora of resources, including CPUs, GPUs, storage, databases, and analytics, etc., over the "cloud." Cloud computing providers manage the hardware and software, creating a flexible and scalable environment that can be customized to meet specific user demands. A vital component of cloud computing is *Infrastructure as a Service* (IaaS) [6, 15, 23, 31], which offers on-demand access to virtualized computing resources on a pay-as-you-go basis, enabling users to run applications independently on the cloud.

As a basic unit falling under the IaaS umbrella, Alibaba Cloud's *Elastic Compute Service*[1] (ECS) accommodates instances with specific hardware configurations, providing an easy-to-operate scaling interface. One of the key operations in ECS is autoscaling, which elastically adds or deletes ECS instances to meet constantly changing user demands [25, 45]. As illustrated in Figure 1, the white bars represent the user demands for ECS instances at different timestamps, and the red line represents the amount of ECS instances provided by the cloud. Traditional autoscaling strategies fall into two extremes. The *conservative* strategy in Figure 1(a) provides a


Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611566

[1] https://www.alibabacloud.com/product/ecs

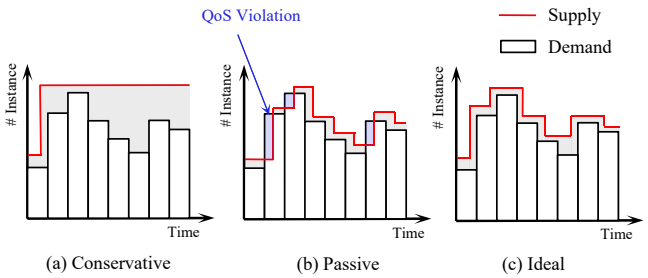



**Figure 1: Three easy-to-understand autoscaling strategies: a) conservative strategy: high costs, low QoS violation; b) passive strategy: low costs, high QoS violation; c) ideal strategy: low costs, low QoS violation.**

safely high enough number of ECS instances based on historical demand statistics to ensure users' demands are always met, but this results in a significant waste of resources, incurring high running costs. On the other extreme, a *passive* strategy shown in Figure 1(b) adjusts the number of ECS instances in a timely manner based on instantaneous user demands, resulting in a decrease in Quality of Service (QoS). This is mainly due to the cold launching of instances—it takes some time to launch new instances to increase the capacity of the cloud.

Predictive autoscaling is closer to the ideal strategy shown in Figure 1(c), which offers an alternative solution by using a forecasting model to predict future demands, thus enabling launching ECS instances in advance to avoid cold launching. Although such researches are being studied [1, 2, 17, 30, 45], they cannot be directly applied to our scenario due to the following three limitations.

**L1: Lack Support to Uncertain Demands.** Existing methods [1, 30, 45] tend to focus on deterministic predictive autoscaling, and their forecasting module may not successfully predict the uncertain demands, which are common in Alibaba Cloud. For example, Figure 2 describes the 25-day demand fluctuation of a specific Cluster-HZ, and it is *highly uncertain*. The reason for these highly uncertain fluctuations is that the cluster-level demand is the sum of all user-level demands (a public cloud cluster provides multi-tenant mode), and these heterogeneous users may have completely inconsistent behavior patterns. This brings great challenges to the implementation of predictive autoscaling. That is, if the high uncertainties are not properly accounted for, the incorrect deterministic demand prediction may lead to wrong decisions of ECS autoscaling, ultimately impacting the QoS. To address this issue, a predictive autoscaling framework that considers uncertainties in both the demand prediction and autoscaling phases would be sensible.

**L2: Ignore Scale-sensitive Dependencies.** Consider the example presented in Figure 2, which examines the minute, hour, and day-level demand variations of the Cluster-HZ in Alibaba Cloud. Obviously, none of them has explicit periodicity, which makes traditional forecasting algorithms based on periodic detection fail to obtain valid periodic information on series of any single scale. Therefore, the predictive autoscaling framework based on a single scale cannot achieve accurate autoscaling. According to this insight, it is meaningful to consider two kinds of temporal dependencies—one is the dependence between scales (inter-scale) and the other is the dependence within a single scale (intra-scale). We call the combination of these two *scale-sensitive dependencies*, which has a great potential to capture the complex fluctuations of uncertainty, thereby improving the accuracy of predictive autoscaling.

**L3: Fail to Adapt to Specific Business Scenario.** Although existing scalers such as [25, 45] have demonstrated significant performance improvements for instance scaling, they do not take into account more specific and complex business scenarios. For example, in Alibaba cloud, each instance may be in the launching, running, and draining phases, each of which takes time and incurs a different cost. Additionally, when an instance is in the draining stage, it can be re-launched immediately. However, most scalers simplify these real-world scenarios when building their formal modeling. In particular business contexts, the occurrence of novel conflicts may arise, particularly in cases where a sudden decline in customer demand is promptly succeeded by an abrupt surge. Should instances be drained or launched in real-time, contingent upon customer demand, the consequential expenses of replacing resources may prove substantial. Conversely, upholding the current number of instances can engender augmented costs associated with resource utilization. Regrettably, the inadequacies of simplified modeling frequently overlook the intricacies inherent to the particular business scenario at hand. To this end, our *MagicScaler* aims to incorporate demand forecasting uncertainties to balance instance cost and QoS by considering these complexity factors in real business scenarios.

**Contributions.** To address the above limitations, we propose a novel predictive autoscaling framework *MagicScaler*, consisting of a Multi-scale attentive gaussian process based predictor and an uncertainty-aware Scaler. First, the predictor carefully bridges the best of two successful prediction methodologies—multi-scale attention mechanisms, which are good at capturing complex, multi-scale features, and stochastic process regression, which is able to quantify prediction uncertainty, thus achieving accurate demand prediction with quantified uncertainty levels. Second, the scaler takes into account the quantified future demand uncertainty into a judiciously designed loss function with stochastic constraints considering the flexible trade-off between running costs and QoS violations. In the end, we list our contributions as follows:

(1) We design a novel multi-scale attentive Gaussian process based predictor to accurately predict future demands with quantified uncertainty. The predictor leverages a two-stage (i.e., internal vs. external) multi-scale feature extraction to capture scale-sensitive temporal dependencies and a Gaussian process on top of the extracted multi-scale features to enable accurate future uncertain demand prediction.
(2) We develop an uncertainty-aware scaler, which internalizes predicted uncertainty via stochastic constraints on different tolerance levels of QoS violations, achieving flexible trade-offs between running costs and QoS violations.
(3) Extensive experiments are reported for both demand prediction and autoscaling, demonstrating our proposal's effectiveness and efficiency.

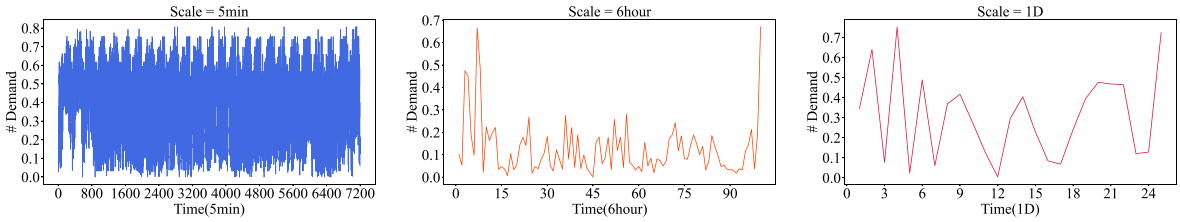



**Figure 2: ECS demand time series of Alibaba Cloud's cluster-HZ at different scales. *Magnitude is eliminated for privacy**

## 2 RELATED WORK

### 2.1 Time Series Forecasting

Various methods for time series analytics, including forecasting, exist, such as statistical methods, deep learning methods, *etc.* [7, 8, 11–14, 19, 20, 32, 39, 43, 44, 46, 50]. For the sake of brevity, we start with the most advanced Transformer-based methods. Transformer [35] models are well suited for time series forecasting since they can capture long-term dependencies in the data, as well as short-term correlations. By using transformer models, predictive scheduling algorithms can make accurate and efficient predictions about future workloads and enable better decisions with resource allocation.

More recently, time-series Transformers [40, 42, 47] were introduced for the forecasting task by leveraging self-attention mechanisms to learn complex patterns and dynamics from time series data. Binh and Matteson [33] propose a probabilistic, nonautoregressive transformer-based model with the integration of state space models. The original quadratic complexity was reduced to $O(LlogL)$ by introducing sparsity into the attention mechanism with the ProbSparse attention of the Informer model [48], and the logSparse attention mechanism. While such attention mechanisms operate on a point-wise basis, Autoformer [22] used a cross-correlation-based attention mechanism to operate at the level of subsequences. Triformer [10] introduces a novel patch attention with linear complexity. When stacking multiple layers of the patch attentions, a triangular structure is proposed such that the layer sizes shrink exponentially, thus maintaining linear complexity. FEDformer [49] employs frequency transform to decompose the sequence into multiple frequency domain modes to extract the feature, further improving the performance of Autoformer. However, there are often certain patterns in public data sets that are easy to mine (such as the laws of weather and the spatiotemporal characteristics of electricity), which makes these state-of-the-art models achieve near-perfect results on them. However, these methods may not achieve satisfactory results when faced with rarely regular and highly uncertain demand sequences.

### 2.2 Predictive Autoscaling

Predictive autoscaling is used to dynamically add or delete computing resources, such as instances, CPU, and memory, in order to closely match the ever-changing computing demand. This technique is essential for efficient resource utilization with satisfactory QoS in cloud computing. Predictive autoscaling algorithms use forecasting models to predict the future workload and make decisions regarding resource allocation and scheduling.

Existing autoscalers generate scaling decisions based on control theory, reinforcement learning, queuing theory, and rule-based methods [4, 5, 9, 24, 27, 41]. Most of these methods either make scaling decisions based on a mean demand estimation without considering uncertainty or handle uncertainty in a heuristic way. For example, model predictive control is adopted for predictive autoscaling in [30] which utilizes ARMA model for workload forecasting and look-ahead controller for resource allocation without considering uncertainty. The autoscaling scheme RobustT2Scale [18] integrates a fuzzy controller with an online learning mechanism that can cope with uncertainties but is not general enough to handle variable periodic patterns [16, 38]. Instead, we directly incorporate workload uncertainty into scaling decisions via stochastic constraints that are expressive of QoS requirements to derive robust scaling decisions.

Recently, Qian et al. proposed a new autoscaling framework called *RobustScaler* [25] that uses non-homogeneous Poisson processes (NHPP) modeling and stochastically constrained optimization to achieve a superior trade-off between instance cost and quality of service (QoS). However, it is important to note that *RobustScaler* is designed for scaling-per-query scenarios, which is different from the scaling-per-cluster scenario in this study. In the context of scaling-per-query scenarios, instances are typically terminated after the completion of a single query, and their utilization is not extended to accommodate subsequent queries. However, our scenario diverges as our instances possess the inherent capacity for reuse across multiple queries. In a similar vein, Xue et al. [45] proposed an end-to-end predictive meta-model-based reinforcement learning (RL) algorithm to maintain stable CPU utilization in the cloud. Their method uses a deep attentive periodic model to predict workload and compute CPU utilization through the Attentive Neural Process. Finally, the model-based RL is used to find the best scaling decider. However, this approach does not account for uncertainties in CPU utilization prediction and the impact of service-level agreements (SLAs), making it unsuitable for deployment in real-world cloud services. Remarkably, both *RobustScaler* and RL-based scalers overlook the complexities of various business scenarios where each instance may be in launching, running, or draining phases with different costs. If an instance is in the draining phase, it can be immediately relaunched. As a result, neither *RobustScaler* nor RL-based scalers are suitable for this scenario, and thus, we did not use them as baseline methods.

## 3 PRELIMINARIES

### 3.1 Problem Definition

We first introduce key concepts in autoscaling.

- **Instance:** An instance is the basic unit of resource scheduling in an ECS. An instance contains a specific hardware

configuration (e.g., 2-core CPU, 4GB memory). Both supply and demand are specified by the number of instances.

- **Demand series:** A time series consists of the number of instances that users demand at each timestamp.
- **Supply series:** A time series consists of the number of instances provided by an ECS at each timestamp.

Based on the above concepts, our problem is defined as follows. At a specific timestamp $t$, given a historical demand time series $\mathbb{D}_{t-H+1:t}$ covering the previous $H$ timestamps, we aim to recommend an optimal supply series $\mathbb{S}_{t+1:t+F}$ over the future $F$ timestamps such that both the resource cost and QoS loss are minimized.

(1) **Resource Cost**: The life cycle of an instance is divided into three phases—*launching, running, and draining*. The instance can only be used during the running phase and is unavailable during the scaling phases, i.e., launching and draining. However, the scaling phases also have costs. Thus, the resource cost of an instance consists of two main components, namely the *running cost* and the *scaling cost*.
(2) **QoS Loss**: When using cloud services, customers often have different QoS requirements specified through Service Level Agreements (SLAs) with the service providers. In this paper, we define QoS Loss as the number of instances that do not meet customer SLA requirements per timestamp.

Based on the above key concepts, we present a comprehensive two-stage problem delineation.

*Probabilistic Demand Forecasting.* Given a historical demand time series at timestamp $t$ with a look-back window size $H$, denoted as $\mathbb{D}_t = \langle d_{t-H+1}, d_{t-H+2}, \cdots, d_t \rangle$, probabilistic demand forecast aims at predicting the demand distribution of the future $F$ timestamps, *i.e.*, $\hat{\mathbb{W}}_t = \langle \hat{w}_{t+1}, \hat{w}_{t+2}, \cdots, \hat{w}_{t+F} \rangle$, where $\hat{w}_{t+i}$ represents the demand distribution at timestamp $t+i$, i.e., the probability distribution of the number of instances requested by users at future timestamp $t+i$. Based on the predicted demand distributions $\hat{\mathbb{W}}_t$, we derive $\hat{\mathbb{D}}_t = \left\langle \hat{d}_{t+1}, \hat{d}_{t+2}, \cdots, \hat{d}_{t+F} \right\rangle$, where $\hat{d}_{t+i}$ is the mean value of distribution $\hat{w}_{t+i}$. $\hat{\mathbb{D}}_t$ is used as predicted deterministic demand.

*Autoscaling.* At timestamp $t$, given the predicted user demand distributions in the future $F$ timestamps, i.e., $\hat{\mathbb{W}}_t = \langle \hat{w}_{t+1}, \hat{w}_{t+2}, \cdots, \hat{w}_{t+F} \rangle$, and the current cloud state $S_t$, e.g., current running instances, etc. (ref. to Section 5 for details), *MagicScaler* aims to find an optimal scaling action $A_t^*$, i.e., launching new instances or closing running instances, such that both the instance costs and QoS loss risks are minimized.

$$AutoScale(\hat{\mathbb{W}}_t, S_t) \rightarrow A_t, \tag{1}$$

$$A_t^* = argmin(InstanceCost(A_t), QoSLoss(A_t)), \tag{2}$$

## 3.2 Gaussian Process Regression

A Gaussian process [29] is a collection of random variables which follow a joint Gaussian distribution. Gaussian process regression is a probabilistic regression model based on Gaussian processes. It learns a mapping function $f(\cdot)$ from a multidimensional point $\mathbf{x}_i$, e.g., the user demands in historical timestamps, to a real value, e.g., the user demand of the future timestamp, along its probability distribution. Here, the probability distribution follows Gaussian distributions. Formally, we have $f(\mathbf{x}_i) \sim \mathcal{N}(\mu_i, \sigma_i^2)$.

Assume that we have $P$ training data $(\mathbf{x}_{1:P}, \mathbf{y}_{1:P})$, where $(\mathbf{x}_p, \mathbf{y}_p)$ represents a specific training data at timestamp $p$ and $1 \le p \le P$. More specifically, $\mathbf{x}_p$ is the user demands in a historical time window from timestamp $p$ back to timestamp $p-H+1$, and $\mathbf{y}_p$ is the user demand at $p+1$, i.e., the demand of the next timestamp. Based on the Gaussian process modeling, we derive the joint distribution of the $P$ training data as:

$$\mathbf{f}(\mathbf{x}_{1:P}) \sim \mathcal{N}(m(\mathbf{x}_{1:P}), \mathbf{K}(\mathbf{x}_{1:P}, \mathbf{x}_{1:P})), \tag{3}$$

where $m(\cdot)$ is the mean function, and $\mathbf{K}$ is the covariance matrix derived from the kernel function $\kappa(\mathbf{x}_i, \mathbf{x}_j)$. A generally used kernel $\kappa$ is the squared exponential kernel: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \theta_f \mathbf{exp}\left(-\left|\frac{(\mathbf{x}_i - \mathbf{x}_j)^2}{2\Theta^2}\right|\right)$ with variance $\Theta^2$, scaled by the parameter $\theta_f$ of the observations.

In the inference phase, we take the above Gaussian joint distribution as a prior. Given user demands in a new window $\mathbf{x}^*$, the predicted demand distribution at the next timestamp is modeled as a posterior distribution:

$$f(\mathbf{x}^*) \sim \boldsymbol{N}(\mu(\mathbf{x}^*), \sigma^2(\mathbf{x}^*)), \tag{4}$$

where

$$\mu(\mathbf{x}^*) = \kappa(\mathbf{x}^*, \mathbf{x}_{1:P})\mathbf{K}(\mathbf{x}_{1:P}, \mathbf{x}_{1:P})^{-1}\mathbf{f}(\mathbf{x}_{1:P})$$

$$\sigma^2(\mathbf{x}^*) = \kappa(\mathbf{x}^*, \mathbf{x}^*) - \kappa(\mathbf{x}^*, \mathbf{x}_{1:P})\mathbf{K}(\mathbf{x}_{1:P}, \mathbf{x}_{1:P})^{-1}\kappa(\mathbf{x}^*, \mathbf{x}_{1:P})^T$$

More specifically, the Gaussian process predicts the demand in the next timestamp by following a Gaussian distribution $\boldsymbol{N}(\mu(\mathbf{x}^*), \sigma^2(\mathbf{x}^*))$. When we need to make predictions for the next $F$ timestamps, we iteratively call the Gaussian process predictor $F$ times in an auto-regressive manner.

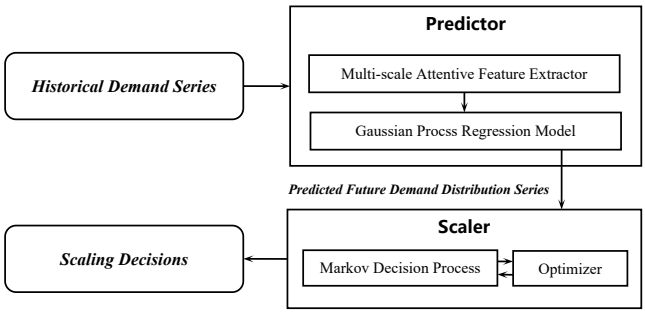



**Figure 3: Framework overview of the proposed *MagicScaler*.**

## 3.3 Overview of Framework

Figure 3 illustrates the overview of the proposal. Specifically, *MagicScaler* consists of two main components—a predictor (described in Section 4) and a scaler (described in Section 5).

(1) **Predictor:** The predictor first employs a multi-scale attentive feature extractor (MAFE) to capture multi-scale features from *historical demand series*. The extracted features, instead of the raw demand time series, are fed into a Gaussian process regression model to predict *future demand distribution series*.

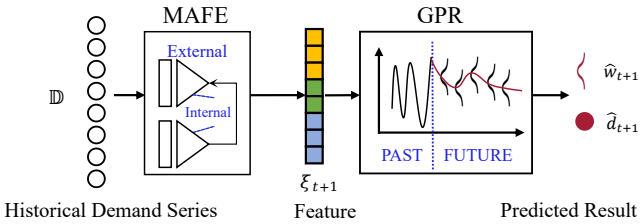



**Figure 4: The overall workflow of the Predictor module.**

(2) **Scaler:** The scaler takes as input the predicted future demand distributions and models the scaling decisions into a Markov decision process. Finally, by considering the instance costs and QoS violation risks, the scaler returns the final *scaling decisions*, i.e., launching new instances or deleting existing instances.

## 4 THE PREDICTOR

Figure 4 describes the overall workflow of the predictor. The input of the predictor is the historical demand series $\mathbb{D}_t$. Multi-scale features $\xi_{t+1}$ are extracted through both internal and external multi-scale attentive feature extractors (MAFE). Then, $\xi_{t+1}$ is fed into the Gaussian process regression (GPR) model, which derives the predicted Gaussian distributions of the next timestamp $t+1$. In the following, we elaborate on the MAFE and GPR modules in details.

### 4.1 Multi-scale Attentive Feature Extractor

Integrating information at different time scales is essential to model and forecast time series accurately. Consider the example presented in Figure 2, which examines the minute, hour, and day-level demand variations of specific Cluster-HZ in Alibaba Cloud. None of them have explicit periodicity, which makes traditional forecasting algorithms based on periodic detection unable to obtain valid periodic information on series of any single scale series. In the context of Section 1, we have showcased the critical characteristic of predictive autoscaling in the cloud computing service, motivating us to improve forecasting accuracy. Therefore, we propose a two-phase Multi-scale Attentive Feature Extractor (MAFE) to capture *scale-sensitive* dependencies, which is mainly divided into an external phase *External-MAFE* and an internal phase *Internal-MAFE*.

*4.1.1 External-MAFE.* We first propose the External-MAFE to capture **inter-scale** dependencies, which improves the stability of the prediction effect via correlations between scales. For example, if the time series at several scales all reflect a traffic peak at a certain time period, a real traffic peak would happen with high likelihood. As shown in Figure 5, External-MAFE takes a raw time series as input and generates its sub-series at different scales, where the raw time series is regarded as the input with the finest scale (marked as white circles) while the other sub-series are down-sampled by the *average* pooling technique (marked as circles with different colors). By refining the extracted features from coarse to fine, the correlation between different scales will be captured.

The External-MAFE starts from the bottom, i.e., the input time series with the coarsest-scale time series (marked as blue). We directly obtain the features extracted by Internal-MAFE (to be detailed in Section 4.1.2, which can be regarded as a black-box function for now). Then, we perform iterative coarse-to-fine steps. At each step, the input time series needs to be fused with the features from the previous step, and then fed into a new Internal-MAFE. Algorithm 1 describes the process of External-MAFE. First, we sort the scale list $L_{scale}$ in descending order (line 1), taking Figure 5 as an example where the sorting result is $\{12, 4, 2, 1\}$. Thus, the first step we process is the sub-series of $scale = 12$. Then, the iterative coarse-to-fine step (lines 2 ∼ 9) is the same as we described before. The fusion operation leverages a fully connected neural network to adjust the dimensionality of the features output from the previous step, and then concatenates with the input of the current step. Note that there is no *fusion* operation in the first step because there is no output from the previous step. Finally, we get the External-MAFE feature $\xi$ of the raw time series.

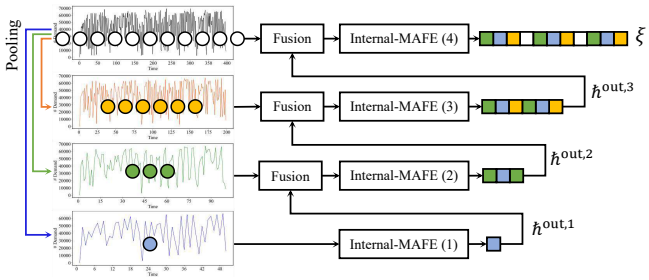


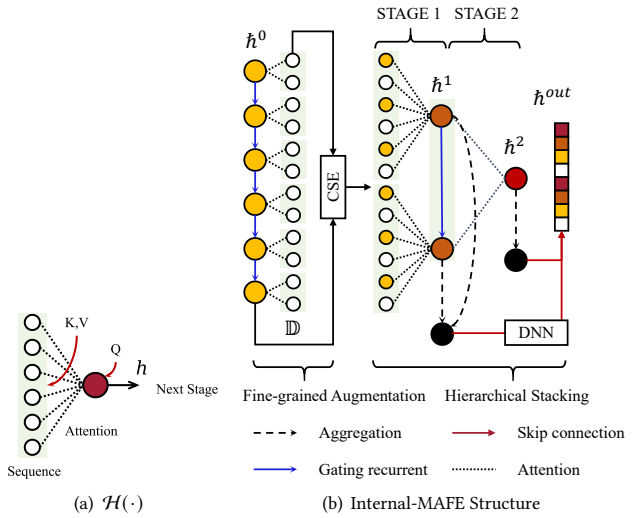

**Figure 5: Design of the External-MAFE module.**

**Algorithm 1:** External-MAFE

**Input:** Sequence $\mathbb{D} = \langle d_1, d_2, \cdots, d_n \rangle$, Scale list $L_{scale}$
**Output:** Feature $\xi$

1 DescendingSort($L_{scale}$);
2 **for** $i \leftarrow 1$ **to** $L_{scale}.length$ **do**
3 $\quad \mathbb{D}^i$ = Pooling($\mathbb{D}, L_{scale}[i]$);
4 $\quad$ **if** $i = 1$ **then**
5 $\quad\quad \hbar^{out,i}$ = Internal-MAFE ($\mathbb{D}^i$);
6 $\quad$ **else**
7 $\quad\quad \mathbb{D}^i$ =Fusion ($\mathbb{D}^i, \hbar^{out,i-1}$);
8 $\quad\quad \hbar^{out,i}$ = Internal-MAFE ($\mathbb{D}^i$);
9 $\quad \xi = \hbar^{out,i}$;
10 Return $\xi$;

*4.1.2 Internal-MAFE.* We propose the Internal-MAFE to capture attentive dependencies in a scale (**intra-scale** dependencies). Unlike External-MAFE, Internal-MAFE internalizes all hidden features of different granularity in one shot. Internal-MAFE takes as input $\mathbb{D}$ of a specific scale determined by External-MAFE and returns the comprehensive coarse-and-fine-grained attentive feature $\hbar^{out}$. The intuition behind this is that in highly uncertain time series, the underlying pattern can only be reflected by the dependence between data of different granularity. In particular, our Internal-MAFE includes *Fine-grained Augmentation* module and *Hierarchical Stacking* module, as shown in Figure 6(b).




(a) $\mathcal{H}(\cdot)$ (b) Internal-MAFE Structure

**Algorithm 2:** Internal-MAFE

**Input:** Sequence $\mathbb{D} = \langle d_1, d_2, \cdots, d_N \rangle$, patch sizes $L_{ps}$
**Output:** Embedding $\hbar^{out}$

1 DescendingSort($L_{ps}$);
2 $\hbar^0 = \mathcal{H}(\mathbb{D}, L_{ps}[-1])$;
3 $\hbar^0$ = CrossScaleEmbedding($\mathbb{D}, \hbar^0$);
4 $\sum_{skip} = 0$;
5 **for** $i \leftarrow 1$ **to** $L_{ps}.length$ **do**
6 $\quad ps = L_{ps}[i]$;
7 $\quad \hbar^i = \mathcal{H}(\hbar^{i-1}, ps)$;
8 $\quad \hat{\hbar}^i$ = Aggregate ($\hbar^i$);
9 $\quad skip$ = Linear ($\hat{\hbar}^i, ps$);
10 $\quad \sum_{skip} = \sum_{skip} + skip$;
11 $\hbar^* = \text{ReLu}(\sum_{skip})$;
12 $\hbar^{out}$ = DNN ($\hbar^*$);

**Figure 6: Design of the Internal-MAFE module.**

*Hidden Feature Extractor.* Before organizing the logic of data flow within the Internal-MAFE, we first elaborate a basic operator, namely *hidden feature extractor*, denoted by $H(\cdot)$. As shown in Figure 6(a), given the input series $D = \langle d_1, d_2, \cdots, d_n \rangle$, we employ the attention mechanism [35] to extract the features of $D$. First, we randomly initialize a learnable feature placeholder $h$ to act as $Q$, then $D = \langle d_1, d_2, \cdots, d_n \rangle$ in the receptive field act as $K$ and $V$. Second, we update $h$ iteratively as each $K$ attends to $Q$ using Equation (5) as follows

$$h = H(h, d) = \left\{ \varphi\left( \frac{h \cdot (d_i \cdot \mathbf{W}_K)^T}{\sqrt{dim}} \right) (d_i \cdot \mathbf{W}_V) \right\}_{i=1}^{n}, \quad (5)$$

where $\varphi(\cdot)$ means *softmax* operator. Such a feature extraction approach differs from pooling-based self-attention, which uses self-attention that requires $d_i$ to be treated as $Q$, so its complexity reaches $O(n^2)$. However, in our solution, only $d_i$ attends to $h$ is required in $H(\cdot)$, thus $H(\cdot)$ just having $O(n)$ complexity. Also, it does not need to add an extra pooling layer to reduce the dimensionality as a feature.

After the basic operator $H(\cdot)$ is introduced, we further introduce the *sequence-level* operator $\mathcal{H}(\cdot)$. Its main idea is to split the sequence into many patches, and then execute $H(\cdot)$ for each patch. Formally, given a patch size $ps$, an input sequence $\mathbb{D} = \langle d_1, d_2, \cdots, d_N \rangle$ can be split into $N/ps$ equal-size patches, each of which performs an $H(\cdot)$ operation. However, there is no interaction between the split patches for a sequence. To compensate for the reduced temporal receptive field and maintain the temporal information flow, we introduce a gating recurrent connection (the blue arrows in Figure 6(b)) to connect the outputs of the patches as

$$h_{i+1} = tanh(\alpha_1 h_i + \beta_1) \odot sigmoid(\alpha_2 h_i + \beta_2) + h_{i+1}, \quad (6)$$

where $\alpha_1, \alpha_2, \beta_1$ and $\beta_2$ are learned parameters for the recurrent gates, and $\odot$ is an element-wise product. Finally, the hidden feature sequence of $\mathbb{D}$ is generated as

$$\hbar = \mathcal{H}(\mathbb{D}, ps) = (h_1, h_2, \cdots, h_{N/ps}). \quad (7)$$

In the following, we describe the two crucial modules in the Internal-MAFE.

*Hierarchical Stacking.* First, we determine the overall hierarchical structure of Internal-MAFE by specifying a list of patch sizes $L_{ps}$. For example, in the right half of Figure 6(b), we specify $L_{ps} = \{6, 2\}$ and the input sequence's size $|\mathbb{D}| = 12$. This means that we will stack a two-stage Internal-MAFE, and the stage 1 uses the largest patch size $ps = 6$ (refer to line 1 of algorithm 2). Then, we obtain the feature sequence $\hbar^1$ of this stage according to the operation $\mathcal{H}(\hbar^0, 6)$, where $\hbar^0$ is derived from $\mathbb{D}$ via *fine-grained augmentation* (lines 2 ~ 3). Next, in stage 2, we repeat the same operation on the feature sequence output by the previous stage, i.e., $\hbar^2 = \mathcal{H}(\hbar^1, 2)$ (lines 5 ~ 8). $\hbar_1, \hbar_2$ imply the attentive feature semantics of $\mathbb{D}$ at different scales, respectively . In order to capture these multi-scale semantics, our intuition is to concatenate all the feature sequences of each stage, and then map to the final representation through a DNN (fully connected neural network) layer (line 12). In the implementation of Internal-MAFE, we first aggregate the feature sequence of this stage into a single one at each stage, and then obtain the shortest path to the final representation through skip connections (lines 9 ~ 11).

*Fine-grained Augmentation.* Continuing with the previous example, when the patch size of the first stage is 6, the fine-grained information will be lost because only one feature extraction operation is done for each patch. However, we cannot directly transform it to $L_{ps} = \{2, 6\}$ either, since this would cause each patch of the first stage to have only a limited receptive field and not be able to extract enough information for the next stage. Therefore, it is intuitive to make up for the aforementioned information loss without influencing the next stage. The specific method is shown in the left half of the Figure 6(b). That is, we select the smallest patch size from the patch size list $L_{ps}$ (It is 2 in this case) and obtain $\hbar^0 = \mathcal{H}(\mathbb{D}, 2)$. Then, $\hbar^0$ will perform a cross-scale embedding (CSE) with the raw input data, similar to the *fusion* operator in External-MAFE.

## 4.2 MAFE based Gaussian Process Regression

Gaussian process regression (GPR) can model uncertainty well, and its Gaussian distribution representation can be obtained for each timestamp of time series forecasting, as described in Section 3.2.

Given the whole demand time series data for training $D = \langle d_1, d_2, \cdots d_K \rangle$, we use sliding window of size $H$ to generate $N$ subsequences $\mathrm{S} = \langle S_1, S_2, \cdots, S_N \rangle$ with , where

$$S_i = \langle d_i, d_{i+1}, \cdots, d_{i+H-1} \rangle .$$

For a subsequence $S_i$, we define its future demand $d_{i+H}$ as its label, and build a temporal correlation between a historical subsequence $S_i$ and a predicted observation $d_{i+H}$ for time series forecasting. In other words, we have $N$ training data $\{S_i, d_{i+H}\}_{i=1}^{N}$ with $d_{i+H}$ as labels. This way, we can use Equation 3 to model the joint distributions.

In our approach, we leverage the learned multi-scale features in a Gaussian process regression. Specifically, instead of using the subsequences $S_i$ derived from demand time series, we let the subsequences go through the MAFE first to obtain the corresponding multi-scale features $MAFE(S_i)$. Then, we obtain training data $\{MAFE(S_i), d_{i+H}\}_{i=1}^{N}$, which is used in Equation 3 to build the joint distributions. GPR achieves the error between the predicted value $\hat{d}_{i+H}$ and the predicted observation $d_{i+H}$ through the loss function, thereby optimizing the parameters in multi-scale attentive feature extractor and kernel functions.

Given a new subsequence $S^*$, inference aims to predict its future demand distribution using Equation 4. Similarly, we use $MAFE(S^*)$ as the input in Equation 4, to derive the future demand distribution.

# 5 THE SCALER

An effective and efficient autoscaling strategy is crucial for the elastic scaling of cloud resources, as it helps to control resource costs and to prevent QoS loss. Figure 7 illustrates the overview of our framework, including probabilistic demand forecasting, MDP (Markov decision process), Optimizer, and Scaling Decision Executor. In particular, our framework takes a probabilistic demand forecasting $\hat{\mathbb{W}}_t$ as input, which enables uncertain awareness. Subsequently, we formulate the user demand scaling problem as Markov decision process to further capture the uncertainties existing in the environment. Next, considering MDP optimization is an infinite Bellman equation optimization problem, we introduce a Receding Horizon optimal framework, which converts the solution of the Bellman equation in an infinite time domain to a stochastic programming solution in a finite horizons, enabling us to find the optimal policy to approximate best solution of the Bellman equation. Finally, based on the policy, our framework conduct scaling planning to balance the instance cost and QoS by taking launching, running, and draining into account for each instance.

## 5.1 Markov Decision Process

Generally, the Markov Decision Process (MDP) can be defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, r, p, \gamma)$, where $\mathcal{S}$ denotes the state space, $\mathcal{A}$ denotes the action space, $p$ denotes the transition probability, $r$ represents the reward, and $\gamma \in [0, 1)$ is the discount factor [45]. To this end, when given $s \in \mathcal{S}$ and $a \in \mathcal{A}$, $r(s, a)$ represents the expected reward, $p(s' \mid s, a)$ is the probability to reach the state $s'$ from state $s$ by taking action $a$. A policy is utilized to select actions in the MDP, which is represented by $\pi$. Based on the intuition behind MDP, we formulate our scaling problem as an MDP. Given the probabilistic demand forecasting (ref. as to Section 4), we aim at learning optimal Scaler (i.e., policy $\pi$) to continually balance the instance cost (i.e., launching, running, and draining cost) and QoS. The goal is to keep low instance cost with low QoS loss.

*5.1.1 MDP for Autoscaling.* Considering the real-world business scenario, we formulate the MDP of scaling strategy as:

**State Space** $\mathcal{S}$: Sate $S_t = (x_t, l_t, sd_{1t}, sd_{2t}, sd_{3t}) \in \mathcal{S}$ represents the state of the Cloud at timestamp $t$, where $x_t \in N^+$ is the number of running instances at timestamp $t$, $l_t$ denotes the number of instances that cannot be provided due to insufficient resource provisioning before time $t$. To clarify, the ECS cloud computing platform has a unique feature where instances can be relaunched during the "draining period" without incurring any additional scaling costs. Therefore, it is important to keep track of these instances and determine the number of instances in the draining phase at timestamp $t$. Typically, the draining phase of an instance lasts for three timestamps. Thus, we use $sd_{1t}, sd_{2t}, sd_{3t}$ to denote the number of instances that are in the 1st, 2nd, and 3rd timestamps of their draining phases, respectively.

**Action Space** $\mathcal{A}$: Given state $S_t$, we define action space $\mathbb{A}_t(S_t)$ as $A_t := \{\lambda_t, \eta_t, \beta_t\} \in \mathbb{A}_t(S_t)$. In particular, at each timestamp $t$, the decision maker should perform the following two actions: (1) to implement our scaling strategy, we make decisions at every timestamp using a binary variable called $\lambda_t$. This variable will take on either a value of 0 or 1 to represent the launching or draining of instances. We also introduce a positive integer variable called $\eta_t$, which will represent the number of instances to be launched or drained at time $t$. To this end, we formulate $\lambda_t \eta_t$ as launching decision, i.e., launching $\lambda_t \eta_t$ new instances, and $(1 - \lambda_t)\eta_t$ as capacity reduction decision, i.e., putting $(1 - \lambda_t)\eta_t$ running instances into the draining phase, at time $t$. (2) we perform a re-scaling decision to recover instances from the draining phase to the running phase, in order to avoid the resource cost of having to repeatedly relaunch. To this end, we define the variable $\beta_t \in N^+$ to represent the number of re-launching decisions, with a constraint $\beta_t \leq sd_{1t} + sd_{2t}$ since the re-scaling decision $\lambda_t$ is to be used at timestamp $t+1$. Therefore, the number of re-scaling instances at time $t$ should be lower than the number of draining instances of $sd_{1t}$ and $sd_{2t}$ at time $t$.

**Cost**: Instance cost involves two key components, i.e., running cost and scaling cost. In contrast, in classic MDP we maximize Reward, but here we minimize cost. To optimize the overall cost, it is important to reduce the running cost, which can be negatively impacted by oversupply and result in additional idle running cost. Therefore, we formulate a cost function at each time $t$ as

$$\begin{aligned} Cost(S_t, A_t) = & \underbrace{W_1 \mathbb{E}[(x_t - (l_t + w_t - \rho)_+)_+]}_{C_1} \\ & + \underbrace{W_2 \lambda_t \eta_t}_{C_2} + \underbrace{W_3 sd_{1t} + W_4 sd_{2t} + W_5 sd_{3t}}_{C_3} \end{aligned} \tag{8}$$

where part $C_1$ denotes the idle resource cost. In particular, the unprocessed user demand $l_t$ and the new user demand $w_t$ made the total resource demand at timestamp $t$. When the total instance

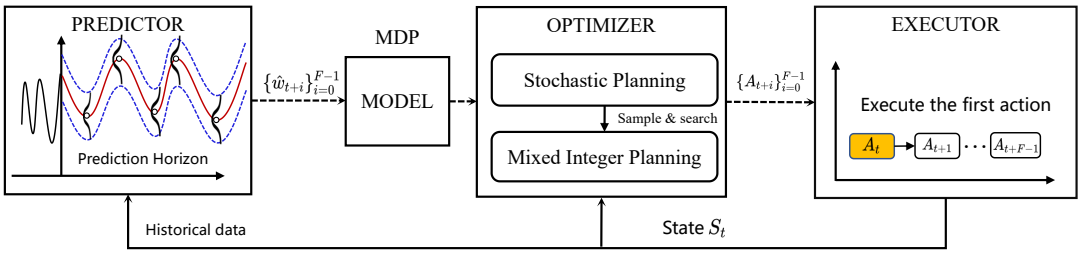



**Figure 7: The overall workflow of the Scaler module.**

demand is less than $\rho$ which denotes the physical resouce, no ECS instance is required. When the total instance demand is greater than $\rho$, the difference between total demand and the amount of physical machine instances is defined as the number of ECS instances that are required at timestamp $t$, which is denoted as $(l_t + w_t - \rho)_+$. Similarly, at timestamp $t$, when the instance supply $x_t$ is less than or equal to $(l_t+w_t-\rho)_+$, no idle cost is incurred, and when $x_t$ is greater than $(l_t + w_t - \rho)_+$, the difference between $x_t$ and $(l_t + w_t - \rho)_+$ is defined as the amount of over-provisioned resources, i.e., the resource idle cost, which is denoted as $(x_t - (l_t + w_t - \rho)+)+$. Since $w_t$ is a probability distribution, the instance idle cost is described as the expectation of $\mathbb{E}[(x_t-(l_t+w_t-\rho)_+)_+]$. In addition, parts $C_2$ and $C_3$ denote the scaling costs. In particular, part $C_1$ use expectation $\mathbb{E}[(x_t - (l_t + w_t - \rho)_+)_+]$, since the predicted user demand $w_t$ is uncertainty, part $C_2$ represents the scaling out cost, and $C_3$ denotes the draining costs at different draining stage. $W_1$, $W_2$, $W_3$, $W_4$, and $W_5$ are the coefficient parameter for each cost.

In addition, we introduce tolerance factor $d$ for a more flexible strategy that balances instance cost and QoS, where $d \in [0, 1]$.

*5.1.2 State transition equation.* When given $A_t := \{\lambda_t, \eta_t, \beta_t\} \in \mathbb{A}_t(S_t)$ at timestamp $t$, the state $S_t = (x_t, l_t, sd_{1t}, sd_{2t}, sd_{3t})$ will be updated to $S_{t+1} = (x_{t+1}, l_{t+1}, sd_{1,t+1}, sd_{2,t+1}, sd_{3,t+1})$ at the timestamp $t + 1$. We formulate the update process as follows

$$x_{t+1} = x_t + \lambda_t\eta_t + (\lambda_t - 1)\eta_t + \lambda_t\beta_t \tag{9a}$$
$$sd_{1,t+1} = (1 - \lambda_t)\eta_t \tag{9b}$$
$$sd_{2,t+1} = (sd_{1t} - \lambda_t\beta_t)_+ \tag{9c}$$
$$sd_{3,t+1} = sd_{2t} - (\lambda_t\beta_t - sd_{1t})_+ \tag{9d}$$
$$l_{t+1} = \mathbb{E}[((l_t + w_t - \rho)_+ - x_t)_+] \tag{9e}$$

where Eq. (9a) describes the update process for available autoscaling resources, Eqs. (9b) to (9d) depict the update process for draining resources at different stages, and Eq. (9e) represents the update process for unmet instance demands.

## 5.2 Optimizer

After formulating the scaling as MDP, the next step is to find the best scaling strategy through the optimizer, including policy learning, value function estimation, and sampling and search.

*5.2.1 Policy learning.* To the best of our knowledge, the policy learning aims to project the state $S_t$ to a set of optimal policies, which is given as $\pi^* = \{A_1^*, A_2^*, ..., A_i^*\} \in \Pi$ with availability of Markovian and deterministic policy $\Pi$. To this end, we minimize the total expectation cost in the infinite domain, which is formulated as

$$Cost^*(S_1) = \min_{\pi\in\Pi} \mathbb{E}[\sum_{t=1}^{\infty} Cost(S_t, A_t^\pi(S_t)|S_1)] \tag{10}$$

where $S_1$ is the initial state of the system. And the corresponding value function of Bellman equation is formulated as

$$V_t(S_t) = \min_{A_t\in\mathbb{A}_\approx(S_t)} \{Cost(S_t, A_t) + \mathbb{E}[V_{t+1}(S_{t+1}|S_t, A_t]\} \tag{11}$$

where $V_t(S_t) = \mathbb{E}[\sum_{t=t'}^{\infty} Cost(S_{t'}A_{t'}|S_t)]$ is the best expected cost of the system starting from time $t$, and the optimal expected cost satisfies $V_1(S_1) = Cost^*(S_1)$. However, to the best of our knowledge, it is very difficult to directly solve the value function in an infinite field [21] to obtain the best policy. To circumvent this, we borrow the intuition behind the forward-looking optimization from Model Predictive Control (MPC) [3] to provide approximated solutions.

*5.2.2 Value Function Estimation.* As mentioned in Section 5.2.1, it is challenging to directly solve the Eq. (11) in the infinite domain. To this end, we use forward-looking optimization (i.e., receding-horizon optimal) to approximate the value function, where we first truncate the infinite domain into finite $F$ horizons and then conduct receding-horizon operation to estimate each value of horizons in finite domain that can be treated as best scaling strategy for the future $F$ steps. Generally, we solve the stochastic planning model in the future finite horizons (i.e., $F$ steps) to approximate the optimal policy at timestamp $t$. Finally, we sum over the value for all horizons, denoted as $V_t(\hat{S}_t)$, as the value for $V_t(S_t)$. However, we just take the first step as our current scaling strategy. Specifically, we have developed a corresponding model as follows

$$\min \sum_{i=0}^{F-1} Cost(t + i)$$
$$s.t. \quad x_{t+1} = x_t + \lambda_t\eta_t + (\lambda_t - 1)\eta_t + \lambda_t\beta_t \tag{12a}$$
$$sd_{1,t+1} = (1 - \lambda_t)\eta_t \tag{12b}$$
$$sd_{2,t+1} = (sd_{1t} - \lambda_t\beta_t)_+ \tag{12c}$$
$$sd_{3,t+1} = sd_{2t} - (\lambda_t\beta_t - sd_{1t})_+ \tag{12d}$$
$$l_{t+1} = E[((l_t + w_{t+1} - \rho)_+ - x_{t+1})_+] \tag{12e}$$
$$\beta_t \le sd_{1t} + sd_{2t} \tag{12f}$$
$$l_t \le E[w_t d] \tag{12g}$$
$$x_t, sd_{1t}, sd_{2t}, sd_{3t}, \beta_t, \eta_t\, w_t \in N^+ \tag{12h}$$
$$\lambda_t \in \{0, 1\} \tag{12i}$$
$$d \in [0, 1] \tag{12j}$$

where Eqs. (12a) to (12e) represent state transition equations for each state variable. Eq. (12f) ensures that each re-launching decision is smaller than the sum of instances currently in the first and second stages of draining. Next, Eq. (12g) limits the unmet ECS demand at each timestamp $t$ to a fixed value that is adjusted by tolerance $d$, which results in at least $100 \cdot (1-d)\%$ satisfactory of ECS demand. In case we want to give priority to QoS loss over instance cost, we can set $d$ to 0, meaning that $l_t$, the number of unmet instances, cannot be greater than 0. By varying the tolerance factor $d$, it is possible to flexibly allow different levels of QoS loss. Finally, Eqs. (12h) to (12j) define the nature of each variable.

*5.2.3 Sampling and Searching.* To approximate optimal policy through solving the stochastic planning model in the future finite horizons (i.e., $F$ steps) at timestamp $t$, we typically use Monte Carlo sampling to convert the random constraint into a linear constraint. However, this can result in many additional variables in the final model, making optimization more complex. Fortunately, the left-hand side of the random constraint (Eq. (12g)) is monotonically decreasing. As $x_t$ increases, the expectation on the left-hand side decreases, so it is possible to convert the stochastic constraint to a linear constraint on $x_t$. We propose an algorithm (see Algorithm 3) that uses sampling and search to approximately solve $\mathbb{E}[((l_{t-1}+w_t-\rho)+-x_t)_+] = \mathbb{E}[w_t d]$, leading to $x_t > x_t^*$.

**Algorithm 3:** Sampling and Searching

**Input:** Sample $w_t^m, m = 1, 2...M, l_{t-1}, d, \rho$
**Output:** Minimum amount of resource provisioning $x_t^*$

Sort $\{l_{t-1}+w_t^m-\rho\}_{m=1}^M$ in descending order as $\{(l_{t-1}+w_t-\rho)^{(1)}, ..., (l_{t-1}+w_t-\rho)^{(M)}\}$;
Let $E=0, K=\frac{1}{M}, x_L=0, x_R=0$ ;
**for** $m \in M$ **do**
  $x_L = (l_{t-1}+w_t-\rho)^{(m)}, x_R = (l_{t-1}+w_t-\rho)^{(m+1)}$;
  $\mathbb{E} = \mathbb{E} + (x_L - x_R)K$;
  **if** $\mathbb{E} \geq \mathbb{E}[w_t d]$ **then**
    $\mathbb{E} = \mathbb{E} - (x_L - x_R)K$;
    $x_t^* = x_L + (d - \mathbb{E})/K$ and break;
  **else**
    $x_t^* = x_R$;
  $K = K + \frac{1}{M}$;

As shown in algorithm 3, we sample $M$ samples $w_t^m, m = 1, 2, 3...M$ using Monte Carlo method. Especially, we use $\sum_{m=1}^M((l_{t-1}+w_t^m-\rho)_+ - x_t)_+/M$ to approximate $\mathbb{E}[((l_{t-1}+w_t-\rho)_+ - x_t)_+]$. Considering $\mathbb{E}[((l_{t-1}+w_t-\rho)_+ - x_t)_+]$ is piecewise linear and monotonically decreasing, its slope fall into the range of $\{l_{t-1}+w_t^m-\rho\}_{m=1}^M$. In this case, we can find the optimal $x_t^*$ when $\mathbb{E}[w_t d]$ appears in the corresponding segment. Due to the time complexity of $O(MlogM)$ in the sampling and sorting stage and linear time complexity in the search stage, the overall time complexity of algorithm 3 is $O(MlogM)$. This allows our method to have highly scalability. Finally, we utilize the mature solvers like SciPy[36] as well as other similar solvers, to obtain the optimal action $A_t^*$.

# 6 EXPERIMENTAL EVALUATION

## 6.1 Evaluation of Forecasting

*6.1.1 Experimental Setup.*

*Datasets.* We consider 3 public datasets and 3 private business dataset with different characteristics: (1) $\mathbf{ETT}_{h_1}, \mathbf{ETT}_{m_1}$ (Electricity Transformer Temperature)[2]: $\mathbf{ETT}_{h_1}$ has observations of every 1 hour and $\mathbf{ETT}_{m_1}$ has observations of every 15 minutes. Each observation consists of 6 power load features, making them 6 variate time series. The train/validation/test data cover 12/4/4 months. (2) **Weather**[3]: This dataset contains local climatological data for nearly 1,600 U.S. locations from 2010 to 2013, where data points are collected every 1 hour. Each data point consists of the target value "wet bulb" and 11 climate features. The train/validation/test data cover 28/10/10 months. (3) **Alibaba Cloud**: The private business data includes the resource demand time series of 3 Alibaba cloud clusters from 3 Chinese cities, Hangzhou (**Cluster-HZ**), Shanghai (**Cluster-SH**), and Beijing (**Cluster-BJ**). All private business data are collected from November 1, 2022 to November 30, 2022, with a sampling frequency of 1 minute. Compared to the aforementioned dataset, this dataset has a much smaller sampling granularity. The train/validation data cover Nov. 1st - Nov. 25th, and the test data cover Nov. 26th - Nov. 30th.

*Baselines.* Since classic models like *ARIMA* and basic *RNN*/*CNN* models perform relatively inferior as shown in [48], we mainly include three state-of-the-art transformer-based models and one probabilistic forecasting model for comparison, i.e., *FEDformer* [49], *Triformer* [10], *Informer* [48] and *GP* as baseline models.

*Metric.* We consider 3 random training and validation setups. The results are averaged over 3 runs. For all datasets, we perform standardization such that the mean of variable is 0 and the standard deviation is 1. We use the following evaluation metrics, i.e., the **MAE** and **MSE** value for forecasting to measure the forecast error. Besides, weighted quantile loss, also known as $\rho$-**risk**, is widely used as a metric for probabilistic time series forecasting. For a given quantile $\rho \in (0,1)$, $q_t^{(\rho)}$ indicates the predicted $\rho$-quantile for $y_t$.

$$\rho-risk = 2\frac{\sum_i \left[\mathbb{I}_{q_t^{(\rho)} \leq y_t}(1-\rho)|q_t^{(\rho)} - y_t| + \mathbb{I}_{q_t^{(\rho)} < y_t}\rho|q_t^{(\rho)} - y_t|\right]}{\sum_i |y_i|} \tag{13}$$

*Experimental settings.* In our experiments, the parameters of MagicScaler's MAFE are set to $L_{ps} = \{8, 3, 2\}, L_{scale} = \{4, 2, 1\}$ by default. For the prediction task in the experiment, MagicScaler defaults to $H = 288$. However, we do not specify the length of $H$ for other methods during the experiment, which is the same setting as the related works [22, 48, 49]. In the end, due to the business requirements of the real scene, we uniformly set the forecast step size $F$ to meet one hour (e.g., if the sampling frequency of the data is 1 minute, then $F$ is 60).

[2]https:// github.com/zhouhaoyi/ETDataset
[3]https://www.bgc-jena.mpg.de/wetter/

**Table 1: Comparison of the MSE and MAE results for our proposed MagicScaler with respective baselines.**

| Methods | *MagicScaler* | | *FEDformer* | | *Triformer* | | *Informer* | | *GP* | |
|---|---|---|---|---|---|---|---|---|---|---|
| Metric | MSE | MAE | MSE | MAE | MSE | MAE | MSE | MAE | MSE | MAE |
| ETT$h_1$ | **0.0110** | **0.0807** | 0.0264 | 0.1259 | 0.0273 | 0.1251 | 0.0297 | 0.1346 | 0.0566 | 0.1391 |
| ETT$m_1$ | **0.0057** | **0.0535** | 0.0061 | 0.0539 | 0.0064 | 0.0551 | 0.0072 | 0.0634 | 0.0102 | 0.0945 |
| weather | 0.0725 | **0.1611** | 0.0855 | 0.2120 | **0.0683** | 0.1639 | 0.0762 | 0.1913 | 0.1040 | 0.2247 |
| Cluster-HZ | **0.0471** | **0.1663** | 0.0958 | 0.1773 | 0.0873 | 0.1699 | 0.0807 | 0.1701 | 0.2554 | 0.3450 |
| Cluster-SH | **0.0363** | **0.1272** | 0.1023 | 0.1628 | 0.1117 | 0.1893 | 0.1616 | 0.2852 | 0.6220 | 0.5305 |
| Cluster-BJ | **0.0539** | **0.1705** | 0.1216 | 0.2162 | 0.1321 | 0.2258 | 0.2875 | 0.3822 | 1.0074 | 0.7510 |

**Table 2: Comparsion of uncertainty quantification results for MagicScaler and GP methods.**

| Methods | *MagicScaler* | | *GP* | |
|---|---|---|---|---|
| Metric | 0.5-risk | 0.9-risk | 0.5-risk | 0.9-risk |
| ETT$h_1$ | **0.0527** | **0.0245** | 0.0832 | 0.0628 |
| ETT$m_1$ | **0.0304** | **0.0154** | 0.0412 | 0.0902 |
| weather | **0.1482** | **0.1011** | 0.1666 | 0.2433 |
| Cluster-HZ | **0.1317** | **0.0964** | 0.7685 | 0.4875 |
| Cluster-SH | **0.1568** | **0.1115** | 0.4009 | 0.2911 |
| Cluster-BJ | **0.1984** | **0.1561** | 0.2333 | 0.1960 |

*6.1.2 Experimental Results and Analysis.* Table 1 and Table 2 summarize the time series evaluation results of all the methods on multiple datasets. The best results are highlighted in boldface.

As shown in Table 1, we can observe that : (i) the forecasting accuracy of *MagicScaler* almost outperforms all the existing state-of-the-art algorithms; (ii) *MagicScaler* outperforms *Triformer*, *FEDformer*, and *Informer* significantly due to its multi-scale feature extractor that can comprehensively capture patterns at different scales, resulting in improved prediction accuracy; (iii) *MagicScaler* outperforms *GP* on MAE by decreasing 64.3% in average, which is attributed to the powerful feature extraction ability of deep neural network; (iv) the outstanding performance of *MagicScaler* on the Alibaba Cloud dataset indicates that our model is better suitable for real-world scenarios compared to other baseline models.

The quantification results of forecasting uncertainty are presented in Table 2 and Figure 8. We can get: (i) *MagicScaler* outperforms *GP* in terms of 0.5-risk and 0.9-risk metrics; (ii) from the visualization in Figure 8, the confidence intervals (marked with blue shades) predicted by *MagicScaler* are capable of covering the real value to a significant extent, and even spike-like points are well covered within the confidence interval.

*6.1.3 Ablation Study.* To validate the effectiveness of the key components of the *MagicScaler*, we conduct an ablation study on the Cluster-HZ dataset for External-MAFE, Internal-MAFE and MAFE, respectively. We could draw a conclusion that all components contribute to the final state-of-the-art results to a certain extent. As shown in Table 3, *MagicScaler* surpasses the model with MAFE removed, demonstrating the effectiveness of MAFE in extracting multi-scale feature information and enhancing model prediction performance. The comparison between External-MAFE and Internal-MAFE shows that External-MAFE is more adept at capturing feature information of diverse scales. It is worth noting that *w/o Internal* does not mean that this component is completely removed,

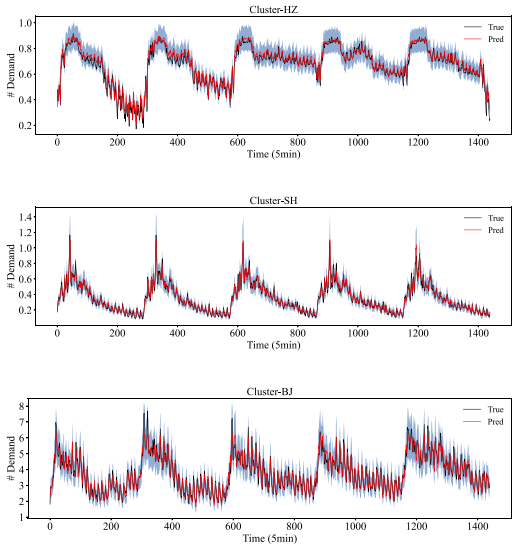



**Figure 8: Visualization of prediction results with uncertainty quantification of MagicScaler. *Magnitude is eliminated**

**Table 3: Ablation study of the proposed MagicScaler on Cluster-HZ datasets.**

| | MSE | MAE | 0.5-risk | 0.9-risk |
|---|---|---|---|---|
| MagicScaler | **0.0471** | **0.1663** | **0.1317** | **0.0964** |
| w/o External | 0.0961 | 0.2188 | 0.1984 | 0.1561 |
| w/o Internal | 0.0679 | 0.1795 | 0.1473 | 0.1225 |
| w/o MAFE | 0.2554 | 0.3450 | 0.7685 | 0.4875 |

but turns its hierarchical structure into a single stage without performing fine-grained augmentation operations. This approach is equivalent to converting Internal-MAFE into an attention-based feature extractor, which serves to remove the component.

*6.1.4 Efficiency.* Figure 9 shows the training time (seconds/epoch) of *MagicScaler* against *FEDformer* and *Informer*, where we compare the average practical efficiencies with 5 runs. When varying $H$: (i) *MagicScaler* is faster than *FEDformer* and *Informer* due to its linear time complexity feature extractor; (ii) *MagicScaler* and *FEDformer* are more stable than Informer. When varying $F$: (i) *MagicScaler* is the fastest in all settings; (ii) the training time of Informer is much longer than of *MagicScaler* and *FEDformer*.

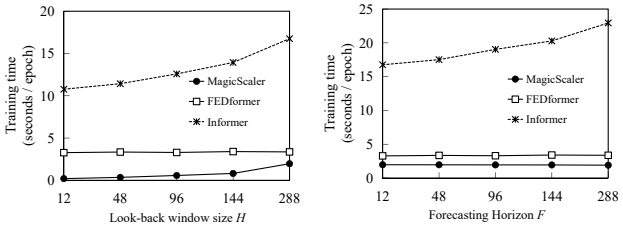



**Figure 9: Comparison of training runtime for MagicScaler with other methods.**

## 6.2 Evaluation of Autoscaling

*Baselines.* In this section, due to the unique nature of the Alibaba Cloud environment, it is difficult to fairly migrate existing autoscaling methods to our scenario, therefore we compare the following scaling methods on real cluster workload data.

- *Passive:* The passive method of scaling is a passive approach that involves making resource launching and draining decisions based on customer demand at each timestamp. While this method may have a lower resource cost, it is also associated with a higher level of QoS loss.
- *Statistics:* This is an autoscaling strategy which utilizes data from the past week at time $t$ to generate statistics, which are then used to forecast future values and perform automatic scaling with a buffer value (similar to the *conservative* strategy described in Section 1).
- *MagicScaler-D: MagicScaler-D* is a deterministic version of *MagicScaler*, which takes as input the mean value of the probabilistic ECS demand forecasting and returns an optimal scaling decision. In particular, we set $F = 12$ and $d = 0$ for *MagicScler-D*, which is meaningful for cloud service providers who prioritize over-provisioning instances to prevent QoS loss.
- *MagicScaler: MagicScaler* takes as input a probabilistic demand forecasting, while keeping the same parameter setting with *MagicScaler-D*.

*6.2.1 Overall Performance Evaluation for Autoscaling.* We compare the performance of *Statistics*, reactive scaling method (*Passive*), *MagicScaler* and its variants on three datasets from Alibaba Cloud (Cluster HZ, SH, and BJ). Figure 10 illustrates the results of our evaluation. The first observation is that, *MagicScaler-D* and *MagicScaler* outperform others in terms of resource cost and QoS loss on Cluster HZ and Cluster SH. Specifically, the QoS loss of *MagicScaler* is significantly reduced with only 11.7% and 4.9% of the QoS loss of Passive and Statistics, by increasing 1.2% and 43.9% resource cost compared to *Passive* and *Statistics*. It is worth noting that although the Statistics has a significant advantage in resource cost metrics on the cluster-SH, this has resulted in a significant QoS loss, which is unacceptable for cloud service providers. It can be seen that *MagicScaler* can balance resource cost and QoS loss well. We also observe that *MagicScaler-D* performs poorly on Cluster BJ, where the resource cost is almost zero on all four days, and the QoS loss is significantly higher than *Passive* and *Statistics*. This is because physical resources in Cluster BJ cover almost all ECS demands and *MagicScaler-D* cannot capture the sudden ECS demand,

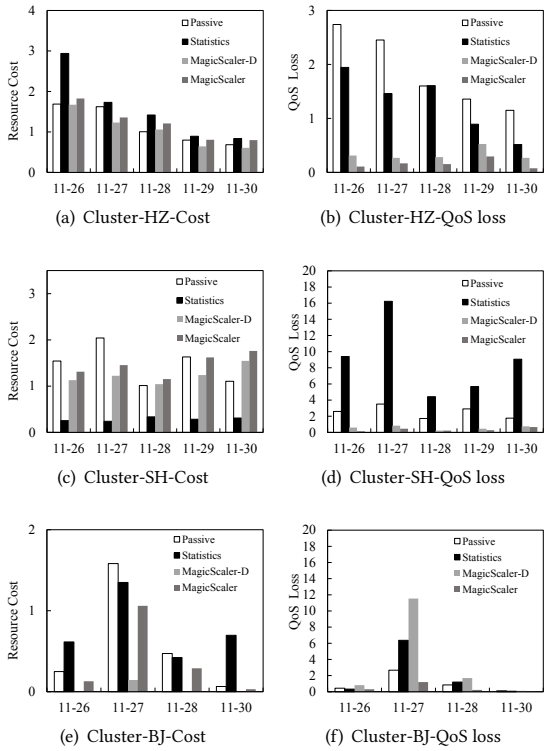



**Figure 10: Comparison of autoscaling performance for MagicScaler with other methods. * The values on the y-axis have been desensitized to protect sensitive information.**

resulting in significant QoS loss. In contrast, *MagicScaler* performs well on Cluster BJ since it can capture sudden ECS demand (i.e., high uncertainties), thus ensuring low QoS loss. Finally, we find that *Statistics* displays poor robustness for different datasets. Although *Statistics* performs comparably well on Cluster-HZ (Figures 11(a) and 11(b)) compared with *Passive*, it shows considerably lower resource cost and significant higher QoS loss in Figures 11(c) and 11(d). This indicates that *Statistics* does not exhibit good robustness for different ECS demand datasets. In contrast, our *MagicScaler* has good robustness across different ECS demand datasets.

*6.2.2 Parameter Sensitivity Analysis.* We further provide experimental insights into the sensitivity analysis of the key parameters (tolerance $d$ and horizons $F$) in the proposed *MagicScaler*.

*Effects of Tolerance d.* To investigate the impact of the tolerance factor $d$ (ref. as to Eq. (12g)), we start $d$ at 0 and increase $d$ by 0.05 each time until 1. In this way, we are able to evaluate the performance of different methods on Cluster-HZ over several days using a Pareto curve, where the left bottom corner is the best scenario to achieve in the Pareto curve, i.e., both low costs and low QoS loss. In particular, we introduce relative QoS and cost as metrics to assess the performance of each approach. Relative QoS is defined as $\text{QoS}_{proposal}/\text{QoS}_{Passive}$, while relative cost is defined as $\text{Cost}_{proposal}/\text{Cost}_{Passive}$. The lower these values, the better the performance. Based on the results reported in Figure 11, we can

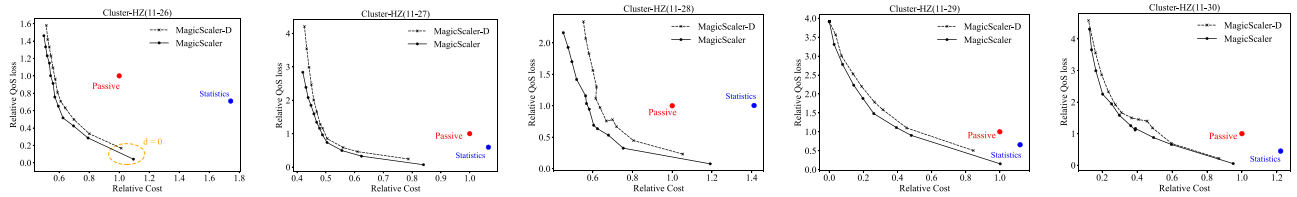



**Figure 11: Tolerance sensitivity experiments of MagicScaler on Cluster-HZ dataset.**

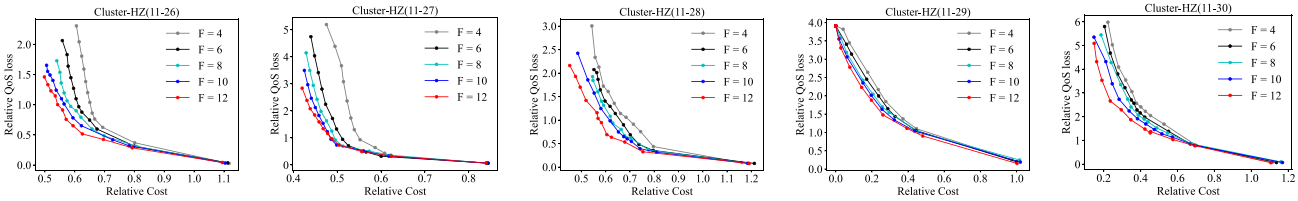



**Figure 12: Horizon sensitivity experiments of MagicScaler on Cluster-HZ dataset.**

obtain: (1) *MagicScaler* performs better than *MagicScler-D* because its Pareto curve is lower and closer to the bottom left corner; (2) both *MagicScaler* and *MagicScaler-D* outperform *Passive* and *Statistics* because they allow for a suitable decision strategy to be found by adjusting $d$; (3) although *MagicScaler* and *MagicScaler-D* have higher instance costs than *Passive*, they are able to achieve the best QoS loss, which indicates their effectiveness in improving the performance of cloud autoscaling.

*Effects of Horizons F.* To study the effect of varying the value of horizons $F$, we study the performance of *MagicScaler* on Cluster-HZ over several days based on Pareto curve, with values of $F$ in $\{4, 6, 8, 10, 12\}$. The results depicted in Figure 12 indicate that the best performance is achieved when $F$ is set to 12, which is used as the default value of our *MagicScaler*. Moreover, we observed that as the value of $F$ increases, *MagicScaler* performs even better. This is because larger $F$ allows *MagicScaler* to gather more information about future trends, enabling it to make more informed scaling decisions. Furthermore, we find that even when $F$ is altered, there exists a range (e.g., [0.6, 0.8] on Cluster-HZ(11-27)) in which *MagicScaler* delivers almost consistent performance across different days. This finding can guide *MagicScaler* to make better scaling decisions while consuming less computation.

*6.2.3 Scalability Analysis of Scaler.* Finally, we evaluated the scalability performance of *MagicScaler* by varying the horizons and the number of samples to measure its running time. Figure 13 presents the experimental results. Our first observation is that the running time of *MagicScaler* increases as the horizon and the number of samples increase. This is because, with a larger horizon, *MagicScaler* needs more time to make decisions on longer sequences. Additionally, with more samples, Monte Carlo simulation takes longer. Our default implementation uses $F = 12$ and *number of samples* $= 100$, which only takes 0.26s to run and meets the QoS requirement of Alibaba cloud service. Furthermore, even when we set $F = 12$ and *number of samples* $= 500$, *MagicScaler* only takes 1.26s to run, which is much less than the decision-making time required by Alibaba

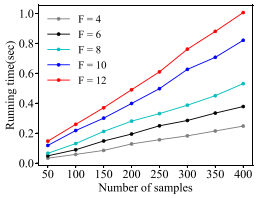



**Figure 13: Running time of MagicScaler for scaling decisions vs. the number of samples.**

cloud service. These results demonstrate the potential for real-time decision-making using our proposed *MagicScaler*.

## 7 CONCLUSION

In this paper, we propose a novel predictive autoscaling framework called *MagicScaler*, which consists of a predictor and a scaler. On one hand, the predictor leverages the strengths of two successful prediction methodologies: multi-scale attention mechanisms, which are effective at capturing complex, multi-scale features, and gaussian process regression, which can accurately quantify prediction uncertainty. On the other hand, the scaler takes into account the quantified future demand uncertainty by using a sophisticated loss function with stochastic constraints, which allows for a flexible trade-off between running costs and QoS violations. Extensive experiments for both demand prediction and autoscaling demonstrate the superior performance of our proposed framework compared to other widely used baseline methods. In particular, the effectiveness of *MagicScaler* has been demonstrated through real-world data from Alibaba Cloud. This suggests that our framework has the great potential to significantly improve the efficiency and effectiveness of autoscaling in cloud computing systems.

## ACKNOWLEDGMENTS

This work was supported by Alibaba Group through Alibaba Innovative Research Program. This work is partially supported by NSFC 62202171 and Shanghai Pujiang Program 21PJ1403200.

# REFERENCES


[1] Muhammad Abdullah, Waheed Iqbal, Josep Lluis Berral, Jorda Polo, and David Carrera. 2022. Burst-Aware Predictive Autoscaling for Containerized Microservices. *IEEE Trans. Serv. Comput.* 15, 3 (2022), 1448–1460.

[2] Muhammad Abdullah, Waheed Iqbal, Abdelkarim Erradi, and Faisal Bukhari. 2019. Learning Predictive Autoscaling Policies for Cloud-Hosted Microservices Using Trace-Driven Modeling. In *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Sydney, Australia, December 11-13, 2019*. IEEE, 119–126.

[3] Abdul Afram and Farrokh Janabi-Sharifi. 2014. Theory and applications of HVAC control systems–A review of model predictive control (MPC). *Building and Environment* 72 (2014), 343–355.

[4] Yahya Al-Dhuraibi, Fawaz Paraiso, Nabil Djarallah, and Philippe Merle. 2017. Elasticity in cloud computing: state of the art and research challenges. *IEEE Transactions on Services Computing* 11, 2 (2017), 430–447.

[5] Ahmed Barnawi, Sherif Sakr, Wenjing Xiao, and Abdullah Al-Barakati. 2020. The views, measurements and challenges of elasticity in the cloud: A review. *Computer Communications* 154 (2020), 111–117.

[6] Dario Bruneo. 2014. A Stochastic Model to Investigate Data Center Performance and QoS in IaaS Cloud Computing Systems. *IEEE Trans. Parallel Distributed Syst.* 25, 3 (2014), 560–569.

[7] David Campos, Tung Kieu, Chenjuan Guo, Feiteng Huang, Kai Zheng, Bin Yang, and Christian S. Jensen. 2022. Unsupervised Time Series Outlier Detection with Diversity-Driven Convolutional Ensembles. *Proc. VLDB Endow.* 15, 3 (2022), 611–623.

[8] David Campos, Miao Zhang, Bin Yang, Tung Kieu, Chenjuan Guo, and Christian S. Jensen. 2023. LightTS: Lightweight Time Series Classification with Adaptive Ensemble Distillation. *SIGMOD* (2023).

[9] Tao Chen, Rami Bahsoon, and Xin Yao. 2018. A Survey and Taxonomy of Self-Aware and Self-Adaptive Cloud Autoscaling Systems. *ACM Computing Surveys (CSUR)* 51, 3 (2018), 1–40.

[10] Razvan-Gabriel Cirstea, Chenjuan Guo, Bin Yang, Tung Kieu, Xuanyi Dong, and Shirui Pan. 2022. Triformer: Triangular, Variable-Specific Attentions for Long Sequence Multivariate Time Series Forecasting. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, Luc De Raedt (Ed.). ijcai.org, 1994–2001.

[11] Razvan-Gabriel Cirstea, Bin Yang, Chenjuan Guo, Tung Kieu, and Shirui Pan. 2022. Towards Spatio- Temporal Aware Traffic Time Series Forecasting. In *ICDE*. 2900–2913.

[12] Razvan-Gabriel Cirstea, Tung Kieu, Chenjuan Guo, Bin Yang, and Sinno Jialin Pan. 2021. EnhanceNet: Plugin Neural Networks for Enhancing Correlated Time Series Forecasting.. In *ICDE*. 1739–1750.

[13] Razvan-Gabriel Cirstea, Bin Yang, and Chenjuan Guo. 2019. Graph Attention Recurrent Neural Networks for Correlated Time Series Forecasting.. In *MileTS19@KDD*.

[14] Christos Faloutsos, Jan Gasthaus, Tim Januschowski, and Yuyang Wang. 2019. Classical and Contemporary Approaches to Big Time Series Forecasting. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19)*. New York, NY, USA, 2042–2047.

[15] Hamid Reza Faragardi, Mohammad Reza Saleh Sedghpour, Saber Fazliahmadi, Thomas Fahringer, and Nayereh Rasouli. 2020. GRP-HEFT: A Budget-Constrained Resource Provisioning Scheme for Workflow Scheduling in IaaS Clouds. *IEEE Trans. Parallel Distributed Syst.* 31, 6 (2020), 1239–1254.

[16] Jingkun Gao, Xiaomin Song, Qingsong Wen, Pichao Wang, Liang Sun, and Huan Xu. 2020. RobustTAD: Robust time series anomaly detection via decomposition and convolutional neural networks. *KDD Workshop on Mining and Learning from Time Series (KDD-MileTS'20)* (2020).

[17] Yisel Gari, David A. Monge, Elina Pacini, Cristian Mateos, and Carlos García Garino. 2021. Reinforcement learning-based application Autoscaling in the Cloud: A survey. *Eng. Appl. Artif. Intell.* 102 (2021), 104288.

[18] Pooyan Jamshidi, Claus Pahl, and Nabor C. Mendonça. 2016. Managing Uncertainty in Autonomic Cloud Elasticity Controllers. *IEEE Cloud Comput.* 3, 3 (2016), 50–60.

[19] Tung Kieu, Bin Yang, Chenjuan Guo, Razvan-Gabriel Cirstea, Yan Zhao, Yale Song, and Christian S. Jensen. 2022. Anomaly Detection in Time Series with Robust Variational Quasi-Recurrent Autoencoders. In *ICDE*. 1342–1354.

[20] Tung Kieu, Bin Yang, Chenjuan Guo, Christian S. Jensen, Yan Zhao, Feiteng Huang, and Kai Zheng. 2022. Robust and Explainable Autoencoders for Unsupervised Time Series Outlier Detection. In *ICDE*. 3038–3050.

[21] Carlos Lagos, Felipe Delgado, and Mathias A Klapp. 2020. Dynamic optimization for airline maintenance operations. *Transportation Science* 54, 4 (2020), 998–1015.

[22] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, and Xifeng Yan. 2019. Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting. In *Annual Conference on Neural Information Processing Systems (NeurIPS 2019), December 8-14, 2019, Vancouver, BC, Canada*. 5244–5254.

[23] Sunilkumar S Manvi and Gopal Krishna Shyam. 2014. Resource management for Infrastructure as a Service (IaaS) in cloud computing: A survey. *Journal of network and computer applications* 41 (2014), 424–440.

[24] Olga Poppe, Qun Guo, Willis Lang, Pankaj Arora, Morgan Oslake, Shize Xu, and Ajay Kalhan. 2022. Moneyball: proactive auto-scaling in Microsoft Azure SQL database serverless. *Proceedings of the VLDB Endowment* 15, 6 (2022), 1279–1287.

[25] Huajie Qian, Qingsong Wen, Liang Sun, Jing Gu, Qiulin Niu, and Zhimin Tang. 2022. RobustScaler: QoS-Aware Autoscaling for Complex Workloads. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2762–2775.

[26] Ling Qian, Zhiguo Luo, Yujian Du, and Leitao Guo. 2009. Cloud computing: An overview. In *Cloud Computing: First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings 1*. Springer, 626–631.

[27] Chenhao Qu, Rodrigo N Calheiros, and Rajkumar Buyya. 2018. Auto-scaling web applications in clouds: A taxonomy and survey. *ACM Computing Surveys (CSUR)* 51, 4 (2018), 1–33.

[28] Aaqib Rashid and Amit Chaturvedi. 2019. Cloud computing characteristics and services: a brief review. *International Journal of Computer Sciences and Engineering* 7, 2 (2019), 421–426.

[29] Carl Edward Rasmussen and Christopher K. I. Williams. 2006. *Gaussian processes for machine learning*. MIT Press.

[30] Nilabja Roy, Abhishek Dubey, and Aniruddha S. Gokhale. 2011. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In *IEEE International Conference on Cloud Computing, CLOUD 2011, Washington, DC, USA, 4-9 July, 2011*. IEEE Computer Society, 500–507.

[31] Nicolas Serrano, Gorka Gallardo, and Josune Hernantes. 2015. Infrastructure as a service and cloud technologies. *IEEE Software* 32, 2 (2015), 30–36.

[32] Jian Tan, Tieying Zhang, Feifei Li, Jie Chen, Qixing Zheng, Ping Zhang, Honglin Qiao, Yue Shi, Wei Cao, and Rui Zhang. 2019. iBTune: Individualized Buffer Tuning for Large-scale Cloud Databases. *Proc. VLDB Endow.* 12, 10 (jun 2019), 1221–1234.

[33] Binh Tang and David S Matteson. 2021. Probabilistic transformer for time series analysis. *Advances in Neural Information Processing Systems* 34 (2021), 23592–23608.

[34] Blesson Varghese and Rajkumar Buyya. 2018. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems* 79 (2018), 849–861.

[35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Annual Conference on Neural Information Processing Systems (NeurIPS 2017) December 4-9, 2017, Long Beach, CA, USA*. 5998–6008.

[36] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. 2020. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature methods* 17, 3 (2020), 261–272.

[37] Ao Wang, Shuai Chang, Huangshi Tian, Hongqi Wang, Haoran Yang, Huiba Li, Rui Du, and Yue Cheng. 2021. FaaSNet: Scalable and Fast Provisioning of Custom Serverless Container Runtimes at Alibaba Cloud Function Compute. In *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*. 443–457.

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

[39] Qingsong Wen, Linxiao Yang, Tian Zhou, and Liang Sun. 2022. Robust time series analysis and applications: An industrial perspective. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*. 4836–4837.

[40] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. 2023. Transformers in time series: A survey. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

[41] Chenggang Wu, Vikram Sreekanti, and Joseph M Hellerstein. 2019. Autoscaling tiered cloud storage in Anna. *Proceedings of the VLDB Endowment* 12, 6 (2019), 624–638.

[42] Neo Wu, Bradley Green, Xue Ben, and Shawn O'Banion. 2020. Deep transformer models for time series forecasting: The influenza prevalence case. *arXiv preprint arXiv:2001.08317* (2020).

[43] Xinle Wu, Dalin Zhang, Chenjuan Guo, Chaoyang He, Bin Yang, and Christian S. Jensen. 2022. AutoCTS: Automated Correlated Time Series Forecasting. *Proc. VLDB Endow.* 15, 4 (2022), 971–983.

[44] Xinle Wu, Dalin Zhang, Miao Zhang, Chenjuan Guo, Bin Yang, and Christian S. Jensen. 2023. AutoCTS+: Joint Neural Architecture and Hyperparameter Search for Correlated Time Series Forecasting. *Proc. ACM Manag. Data* 1, 1 (2023), 97:1–97:26.

[45] Siqiao Xue, Chao Qu, Xiaoming Shi, Cong Liao, Shiyi Zhu, Xiaoyu Tan, Lintao Ma, Shiyu Wang, Shijun Wang, Yun Hu, Lei Lei, Yangfei Zheng, Jianguo Li, and James Zhang. 2022. A Meta Reinforcement Learning Approach for Predictive Autoscaling in the Cloud. In *KDD '22: The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 14 - 18, 2022*. ACM, 4290–4299.

[46] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2020. Effective Travel Time Estimation: When Historical Trajectories over Road Networks Matter. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*. 2135–2149.

[47] George Zerveas, Srideepika Jayaraman, Dhaval Patel, Anuradha Bhamidipaty, and Carsten Eickhoff. 2021. A transformer-based framework for multivariate time series representation learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2114–2124.

[48] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. 2021. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*. 11106–11115.

[49] Tian Zhou, Ziqing Ma, Qingsong Wen, Xue Wang, Liang Sun, and Rong Jin. 2022. FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, Vol. 162. PMLR, 27268–27286.

[50] Yihong Zhou, Zhaohao Ding, Qingsong Wen, and Yi Wang. 2022. Robust Load Forecasting towards Adversarial Attacks via Bayesian Learning. *IEEE Transactions on Power Systems* (2022).