Wasserstein Adversarial Transformer for Cloud Workload Prediction

Shivani Arbat[1], **Vinodh Kumaran Jayakumar**[2], **Jaewoo Lee**[1], **Wei Wang**[2], **In Kee Kim**[1]

[1]University of Georgia

[2] The University of Texas at San Antonio

sga64681@uga.edu, rvn028@my.utsa.edu, jaewoo.lee@uga.edu, wei.wang@utsa.edu, inkee.kim@uga.edu

Abstract

Predictive Virtual Machine (VM) auto-scaling is a promising technique to optimize cloud applications' operating costs and performance. Understanding the job arrival rate is crucial for accurately predicting future changes in cloud workloads and proactively provisioning and de-provisioning VMs for hosting the applications. However, developing a model that accurately predicts cloud workload changes is extremely challenging due to the dynamic nature of cloud workloads. Long-Short-Term-Memory (LSTM) models have been developed for cloud workload prediction. Unfortunately, the state-of-the-art LSTM model leverages recurrences to predict, which naturally adds complexity and increases the inference overhead as input sequences grow longer. To develop a cloud workload prediction model with high accuracy and low inference overhead, this work presents a novel time-series forecasting model called WGAN-gp Transformer, inspired by the Transformer network and improved Wasserstein-GANs. The proposed method adopts a Transformer network as a *generator* and a multi-layer perceptron as a *critic*. The extensive evaluations with real-world workload traces show WGAN-gp Transformer achieves $5\times$ faster inference time with up to 5.1% higher prediction accuracy against the state-of-the-art approach. We also apply WGAN-gp Transformer to auto-scaling mechanisms on Google cloud platforms, and the WGAN-gp Transformer-based auto-scaling mechanism outperforms the LSTM-based mechanism by significantly reducing VM over-provisioning and under-provisioning rates.

Introduction

Resource provisioning and VM auto-scaling of cloud resources are essential operations to optimize cloud costs and the performance of cloud applications (Shen et al. 2011; Mao and Humphrey 2011; Rzadca et al. 2020). Auto-scaling dynamically performs scale-out and scale-in operations as application workloads fluctuate. *e.g.,* change in user requests to the applications. The scale-out operation increases the number of VMs for running the cloud application as the workload increases so that the cloud application leverages enough amount of VMs and meets its performance goal. On the other hand, the scale-in operation automatically downsizes the number of existing VMs by terminating idle VMs when the workload decreases and helps the cloud application minimize the cloud cost.

While auto-scaling offers benefits to cloud applications, unavoidable delays occur during the VM scaling processes, *e.g.,* VM startup delay due to the reactive nature, hence offering suboptimal cloud resource management (Mao and Humphrey 2012; Hao et al. 2021a,b). To address the reactive nature of auto-scaling, predictive auto-scaling approaches have been deeply investigated (Shen et al. 2011; Roy, Dubey, and Gokhale 2011; Calheiros et al. 2015; Kim et al. 2016, 2018; Jayakumar et al. 2020). Predictive auto-scaling mechanisms commonly have two components: *workload predictor* and *auto-scaling module*. The role of workload predictor is to forecast the changes in user requests (or workloads) to cloud applications. An essential step in designing the workload predictor is to understand job arrival rates[1] (JARs).

There have been several approaches used in the workload predictor to model the JARs — for example, statistical time-series methods (Roy, Dubey, and Gokhale 2011; Calheiros et al. 2015), traditional machine learning (Kim et al. 2016; Cortez et al. 2017), ensemble learning (Loff and Garcia 2014; Jiang et al. 2011; Kim et al. 2018, 2020), and deep learning (Kumar et al. 2018; Jayakumar et al. 2020). The state-of-the-art approach in predicting cloud workloads employs a combination of LSTM and Bayesian optimization (Jayakumar et al. 2020) which specifically leverage the power of LSTM to understand JARs using recurrences (Hochreiter and Schmidhuber 1997). However, these recurrences increase complexity and computational overhead as input sequences grow longer. Additionally, while LSTM is capable of detecting long-term seasonality in the cloud workloads, the majority of real-world cloud workloads have random and dynamic burstiness (Islam, Venugopal, and Liu 2015) as shown in cloud workload traces in Figure 1. The sudden spikes in cloud workload traces show unprecedented changes in user request patterns over time. Therefore, there is an urgent need to develop a novel cloud workload predictor to model such dynamic fluctuations and provide high prediction accuracy and low computational overhead.

In this work, we present a novel time-series forecasting model for cloud resource provisioning, called WGAN-gp (Wasserstein Generative Adversarial Network with gradient penalty) Transformer. WGAN-gp Transformer is inspired by Transformer networks (Vaswani et al. 2017) and improved WGAN (Wasserstein Generative Adversarial Networks) (Arjovsky, Chintala, and Bottou 2017). Our proposed model uses the transformer architecture for the *generator* and the *critic* is a multi-layer perceptron (MLP). WGAN-gp Transformer also employs MADGRAD (Mo-

[1] We use the terms workloads, user requests, and job arrival rates interchangeably.

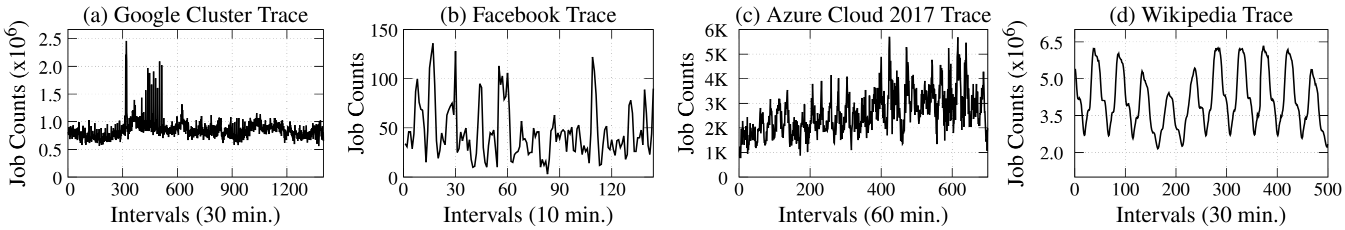

Figure 1: Cloud Workload Traces. (a) Google Cluster Trace, (b) Facebook Hadoop Trace, (c) Azure Cloud 2017 VM Trace, and (d) Wikipedia Trace

mentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization) as the model optimizer for both *generator* and *critic* to achieve better convergence compared to widely adopted Adam optimizer (Defazio and Jelassi 2021). WGAN-gp Transformer is publicly available at https://github.com/shivaniarbat/wgan-gp-transformer.

We thoroughly evaluate the accuracy and overhead of WGAN-gp Transformer on the representative cloud workload datasets, and the evaluation results confirm that WGAN-gp Transformer consistently performs better, yielding lower prediction errors against the state-of-the-art LSTM model (Jayakumar et al. 2020). WGAN-gp Transformer achieves up to 5.1% lower prediction error and $5\times$ faster prediction time against the baseline.

Related Work

Statistical and Machine Learning (ML) Approaches Various methods have been applied to develop workload predictors in predictive VM auto-scaling. The statistical and ML models include Exponential Smoothing, Weighted Moving Average, Autoregressive models (AR) and variations (*e.g.,* ARMA, ARIMA), Support Vector Machine, Random Forest, Gradient Boosting, and others (Shen et al. 2011; Wood et al. 2008; Roy, Dubey, and Gokhale 2011; Calheiros et al. 2015; Yadwadkar, Ananthanarayanan, and Katz 2014; Lin et al. 2015; Cortez et al. 2017; Rzadca et al. 2020). While such statistical and ML approaches could model time-series cloud workloads with cyclic or seasonal trends, such approaches appeared to make sub-optimal predictions for cloud workloads as they were continuously and dynamically changing over time. Moreover, a model can be effective on one "*known*" type of workload, but it often fails to accurately predict future changes in other "*previously unknown*" (previously not trained) workload patterns (Kim et al. 2016).

Neural Networks (NN)-based Approaches The applications of NNs in time-series forecasting can provide improved accuracy across multiple domains. NN models learn to encode relevant historical information from time-series data into intermediate feature representation to make a final forecast with series of non-linear NN layers (Lim and Zohren 2021). For cloud workloads, LSTM (Hochreiter and Schmidhuber 1997) and its variations are studied to forecast the resource demands or user requests. (Nguyen, Klein, and Elmroth 2019; Kumar et al. 2018; Jayakumar et al. 2020). In particular, Jayakumar et al. (Jayakumar et al. 2020) proposed LoadDynamics, a self-optimized generic workload prediction framework for cloud workloads. LoadDynamics performs autonomous self-optimization operations using Bayesian Optimization to repeatedly find the proper hyperparameters in LSTM for handling dynamic fluctuations of cloud workloads. However, LSTM intrinsically depends on capturing long/short dependencies using recurrences. As the input sequence length grows, it increases the complexity of processing such longer input sequences.

Due to the recent advancement of in the field of natural language processing, attention mechanism in Transformer network can be an alternative to recurrences or convolutions (Vaswani et al. 2017). For example, TransGAN (Jiang, Chang, and Wang 2021) proposed a strict, Transformer-based GAN, which employs Transformer as both generator and discriminator. For time-series data, the attention mechanism in Transformer network allows the model to focus on temporal information in the input sequences (Lim and Zohren 2021). Moreover, Adversarial Sparse Transformer (AST) (Wu et al. 2020) was proposed to leverage a sparse attention mechanism for increasing the prediction accuracy at the sequence level. This approach employed sparse transformer as generator and MLP as discriminator. Unfortunately, AST still has limitations to accurately predict highly dynamic cloud workloads because AST often loses long-term forecasting information due to the difficulty of training GANs with sparse point-wise connections.

On the other hand, to effectively predict dynamic cloud workloads with capturing long-term temporal information, our method proposes to train the Transformer network using an improved WGAN-gp algorithm (Gulrajani et al. 2017).

Background

Problem definition A univariate time-series is defined as a sequence of measurements of the same variable collected over time. We study univariate time-series data of JARs or user requests rates collected at regular time intervals from various cloud workloads. Let $\mathbf{x} = [x_1, x_2, \ldots, x_T]$ denote a univariate time-series of length T, where $x_t \in \mathbb{R}$ is its value at time t. We use boldface roman lower and upper case letters to denote vectors and matrices, respectively. $\mathbf{x}_{k:\ell}$ denotes the entries with indices from k to ℓ. To prepare a training dataset $\mathbf{X} \in \mathbb{R}^{N \times S}$, $\mathbf{x}_{1:T}$ is split into N time series of length S. We write $\mathbf{x}_{i,1:S}$ to denote the i^{th} time series in $\mathbf{X}$.

Generative Adversarial Networks Generative Adversarial Networks (GAN) (Goodfellow et al. 2014) are *adversarial nets* framework, which simultaneously trains two completing models; a *generative* model (G) and a *discriminative* model (D). The training strategy leverages a min-max game between two competing models (G and D), and the value function $V(D, G)$ is defined as

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim\mathbb{P}_r}[\log D(x)] + \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[\log(1 - D(\tilde{x}))], \tag{1}$$

where $\mathbb{P}_r$ is the real data distribution, $\mathbb{P}_g$ is the model distribution implicitly defined by $\tilde{x} = G(z)$, and $z \sim p(z)$ is a latent variable having a simple distribution such as uniform distribution or standard normal distribution.

Wasserstein GAN Arjovsky et al. (Arjovsky, Chintala, and Bottou 2017) showed that the gradient of Jensen-Shannon divergence used in the original GAN is not smooth and may not be well defined when the model distribution and the true distribution have different supports. To mitigate the issue, Arjovsky et al. proposed Wasserstein GAN (WGAN), which consists of a generator and a critic. In WGAN, the role of the generator is to generate a sample as in the original GAN, while that of the critic is to approximate the Wasserstein distance between $\mathbb{P}_r$ and $\mathbb{P}_g$. The objective function of WGAN is formulated by

$$\min_G \max_{D\in\mathcal{D}} \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] - \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})], \tag{2}$$

where $\mathcal{D}$ is a set of 1-Lipschitz functions, $\mathbb{P}_r$ represents the data distribution, and $\mathbb{P}_g$ is the model distribution implicitly defined by $\tilde{x} = G(z)$ with a latent variable $z \sim p(z)$.

In (Arjovsky, Chintala, and Bottou 2017), the Lipschitz constraint on $\mathcal{D}$ is enforced by clipping the weight values into a small interval $[-c, c]$. However, it is unknown how to choose the hyperparameter c that has a significant impact on the training of WGAN. Furthermore, irregular value surfaces are generated due to hard clipping of the magnitude of each weight. Other weight constraints (*e.g.,* L2 norm clipping, weight normalization) and soft constraints (*e.g.,* L1 and L2 weight decay) also lead to similar problems (Gulrajani et al. 2017).

Gradient Penalty To address the drawbacks of weight clipping, Gulrajani et al. (Gulrajani et al. 2017) proposed an alternative way to enforce Lipschitz constraint. From the observation that functions f with $\|f(x)\| \leq L$ are L-Lipschitz, they proposed to add a penalty term that forces the gradient norm of critic to stay close to 1, which results in the following objective function.

$$L = \mathbb{E}_{\widetilde{x}\sim\mathbb{P}_g}[D(\widetilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)] + \lambda\mathbb{E}_{\overline{x}\sim\mathbb{P}_{\overline{x}}}[(||\nabla_{\overline{x}}D(\overline{x})||_2 - 1)^2], \tag{3}$$

where $\overline{x} = \epsilon x + (1-\epsilon)\widetilde{x}$ and $\epsilon \sim U[0, 1]$. The gradient penalty coefficient λ controls how strictly the constraint is. In our implementation, we set $\lambda = 10$ (the default values suggested by the authors of WGAN-GP).

Wasserstein Adversarial Transformer

As discussed in the Introduction, cloud workloads are often bursty and dynamically fluctuating over time. To accurately forecast future changes in cloud workloads, we propose WGAN-gp Transformer, based on the Transformer network and WGAN.

Given an input time series $\mathbf{x}_{i,1:S}$, $i = 1, \ldots, N$, our model $G(\cdot\,; \boldsymbol{\theta})$ with parameter $\boldsymbol{\theta}$ predicts the values for time steps in $[t_0 + 1, t_0 + \tau]$ conditioning on $\mathbf{x}_{i,1:t_0}$, where $t_0 + \tau = S$. τ is the number of time steps $G(\cdot\,; \boldsymbol{\theta})$ is trained to predict. That is, $\widehat{\mathbf{x}}_{t_0+1:t_0+\tau} = G(\mathbf{x}_{1:t_0}; \boldsymbol{\theta})$. The time ranges $[1, t_0]$ and $[t_0 + 1 : S]$ are referred to as conditioning range and prediction range, respectively.

Architecture of WGAN-gp Transformer Figure 2 illustrates the architecture of the proposed WGAN-gp Transformer. The *critic* network is composed of 3 fully connected layers with `LeakyReLU` as an activation function (Xu et al. 2015; Wu et al. 2020). The *generator* is based on the encoder-decoder architecture of Transformer network, which is composed of one layer of an encoder and subsequent one layer of a decoder. The encoder encodes the input time series $\mathbf{x}_{i,1:t_0}$ into the latent vector $\mathbf{h}_{1:t_0}$. The latent vector $\mathbf{h}_{1:t_0}$ serves as a memory for the decoder to generate values in the prediction range. Transformer networks are order-agnostic (Vaswani et al. 2017; Tsai et al. 2019) and, unlike recurrences and convolutions, it does not have implicit mechanisms to retain temporal (positional) dependency information. To embed the temporal information, positional encoding (Vaswani et al. 2017) is applied to the input $\mathbf{x}_{i,1:t_0}$. Similarly, positional encoding is applied to the input to decoder. The input to decoder is $\mathbf{x}_{i,t_0}$, which is the last time step value of $\mathbf{x}_{i,1:t_0}$.

Adversarial Training The generator (the Transformer network) attempts to generate high quality predictions that look similar to original time series by minimizing the mean absolute error $\mathcal{L}_{\text{MAE}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=t_0+1}^{S}|\widehat{x}_{i,t} - x_{i,t}|$. The objective function for generator is given by

$$L_G = \frac{1}{N}\sum_{i=1}^{N}\|\widehat{\mathbf{x}}_{i,t_0+1:S} - \mathbf{x}_{i,t_0+1:S}\|_1 - \frac{1}{N}\sum_{i=1}^{N} D([\mathbf{x}_{i,1:t_0} \oplus \widehat{\mathbf{x}}_{i,t_0+1:S}])\,, \tag{4}$$

where $[\mathbf{a} \oplus \mathbf{b}]$ denotes the concatenation of two vectors.

`LeakyReLU` is computed by $f(x) = \max(\alpha x, x)$, where $\alpha = 0.2$. The *critic*'s objective function is given as follows.

$$L_C = \frac{1}{N}\sum_{i=1}^{N}\Big\{D([\mathbf{x}_{i,1:t_0} \oplus \widehat{\mathbf{x}}_{i,t_0+1:S}])\Big\} - \frac{1}{N}\sum_{i=1}^{N} D(\mathbf{x}_{i,1:S}) + \lambda\frac{1}{N}\sum_{i=1}^{N}\left((\|\nabla_{\overline{x}_i} D(\overline{x}_i)\|_2 - 1)^2\right)\,, \tag{5}$$

where $\overline{x}_i = \epsilon\,\mathbf{x}_{i,1:S} + (1-\epsilon)[\mathbf{x}_{i,1:t_0} \oplus \widehat{\mathbf{x}}_{i,t_0+1:S}]$ and $\epsilon \sim U[0, 1]$.

Algorithm 1 describes the training of WGAN-gp Transformer, and it is improved over the original proposal (Gulrajani et al. 2017) to train the *generator*. The algorithm updates the *critic* first and then updates the *generator* with learning results from the *critic*. The *generator* is trained with the updated loss function shown in Equation 4 and the *critic* is trained with the updated loss function expressed in Equation 5. We employ MADGRAD optimizer to minimize the

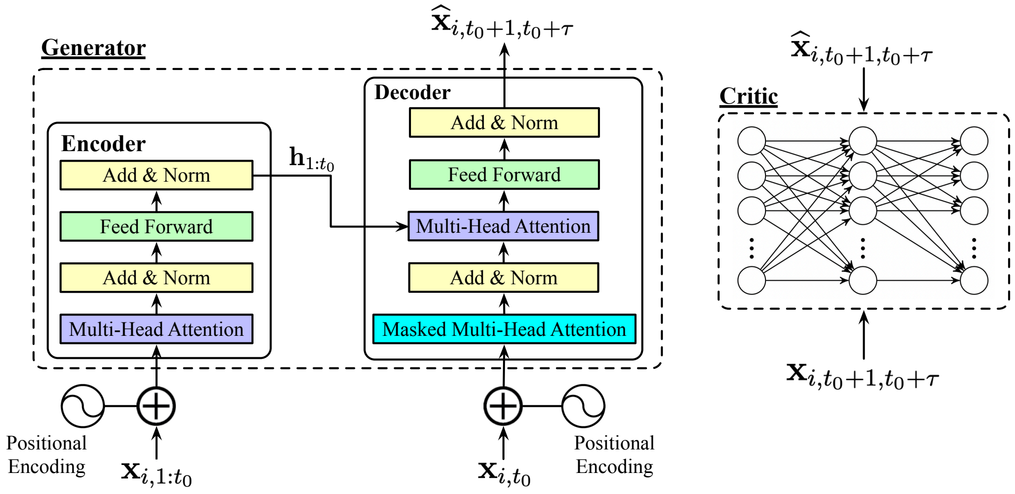

Figure 2: The architecture of WGAN-gp Transformer

Algorithm 1: Training Algorithm of WGAN-gp Transformer

Require: λ, the gradient penalty coefficient. m, the batch size. n_{critic}, the number of iterations of the critic per generator iteration. $m_{MADGRAD}$, momentum value. α learning rate

Require: initial critic parameters w_0, initial generator parameters θ_0.

```
while $\theta$ has not converged do
    for $t = 1, ...., n_{critic}$ do
        for $i = 1, ...., m$ do
            Sample real data
            Sample generator output
            Compute $L_C$
        end for
        $w \leftarrow \texttt{MADGRAD}(\nabla_w \frac{1}{m} \sum_{j=1}^{m} L_C, w, \alpha, m_{MADGRAD})$
    end for
    Sample a batch of generator output
    $\theta \leftarrow \texttt{MADGRAD}(\nabla_\theta \frac{1}{m} \sum_{j=1}^{m} L_G, \theta, \alpha, m_{MADGRAD})$
end while
```

updated loss function with gradient penalty for both *generator* and *critic*. MADGRAD is based upon the dual averaging formulation of AdaGrad for the model optimization (Defazio and Jelassi 2021).

Experiments Setup

Workload Datasets Seven cloud workloads collected from different application categories are used to evaluate WGAN-gp Transformer. The workload traces are described in Table 1. Facebook (Chen et al. 2011), Alibaba (2018)[2], and Google (Wilkes 2011) traces are from data center workloads. Wikipedia workloads are from Wikibench[3]. Three Azure workloads[4] (Azure-VM-2017, Azure-VM-2019, Azure-Func-2019) are from cloud VM and function (serverless) services. These seven workloads are chosen because they have unique characteristics and dynamics in the workload patterns. For example, the data center workloads like Facebook and Google show dynamic spikes and high fluctuations in JARs. Wikipedia dataset represents the behavior of web applications and strong seasonal patterns. Cloud VM and function workloads from Azure show a mixture of characteristics having high fluctuations and seasonality. Among seven selected workloads, Facebook, Google, Azure-VM-2017, and Wikipedia workloads are shown in Figure 1. We omit figures of the other three workload patterns due to the page limitation.

Workload	**Dataset Type**	**Time Interval (in $mins$)**
Facebook	Data Center	5, 10
Alibaba-2018		
Google		10, 30
Wiki	Web	10, 30
Azure-VM-2017	Cloud	10, 30, 60
Azure-VM-2019		
Azure-Func-2019		30, 60

Table 1: Cloud workload datasets

Different time granularities can exhibit subtle variations in the time-series workload patterns. Thus, with seven selected workloads, we generate 15 different *workload configurations* with different time intervals described in Table 1. We use 5 and 10 minutes of the time interval for Facebook and Alibaba workloads. 10 and 30 minutes of the time interval are used for Google and Wikipedia workloads. 10, 30, and 60 minutes of the time interval are used for two Azure-VM workloads. Azure-Func-2019 workload uses two-time interval configurations with 30 and 60 minutes.

Implementation WGAN-gp Transformer is implemented by using `PyTorch` and `scikit-learn`. When training the generator and critic, we use the following configurations; $n_{critic} = 5$, $\lambda = 10$, and α (learning rate) $= 0.001$. We set the length of prediction range to $\tau = 1$ (*i.e.,* one-step

[2] https://github.com/alibaba/clusterdata

[3] http://www.wikibench.eu/

[4] https://github.com/Azure/AzurePublicDataset/

Workload	**History Len.** (n)	**Batch Size** (m)	d_{model}	n_{head}
Facebook	[3-46]	[16-256]	[8, 16, 32, 64, 128, 512]	[4, 8]
Alibaba-2018	[20-324]	[16-1024]		
Google	[28-676]			
Wikipedia (Wiki)	[12-274]			
Azure-VM-2017	[14-682]			
Azure-VM_2019	[14-230]			
Azure-Func-2019	[7-108]	[16-512]		

Table 2: Hyperparameter search space for *generator* in WGAN-gp Transformer

ahead forecasting). The MADGRAD optimizer employs the following default configurations; m (momentum value) $= 0.9$, $weight\_decay = 0$, and $eps = 1e-6$. And the training is performed with 1000 epochs, which works well for our proposed method.

When training and testing WGAN-gp Transformer, we divide the workload dataset sequence into three sub-datasets, containing the first 60%, the next 20%, and the remaining 20%, which are used for the model training, cross-validation, and testing, respectively. We apply a sliding window approach to prepare the input data to WGAN-gp Transformer to divide the data into sequences of (history) length n. The model is trained to predict JAR at the next time step; thus, the sliding window moves with a stride of one-time step to acquire the input sequences. Finally, WGAN-gp Transformer is trained and evaluated on NVIDIA GeForce RTX 2080 Ti GPU machines.

Hyperparameters We use an effective grid search to find the optimal parameters for training WGAP-gp Transformer. We use the configurations described in Table 2 for the hyperparameter search. n (history length) is the length of the input sequence to the model, m is batch size, d_{model} is the number of input features for the encoder and decoder, and n_{head}: the number of heads in multi-head attention layer in encoder and decoder. Please note that the model size (d_{model}) used for training the generator is the same number of input features in the critic linear layers.

Evaluation metric We use Mean Absolute Percentage Error (MAPE) as the accuracy metric to assess the proposed method against the baseline. MAPE is calculated as, $100 \times \left(\frac{1}{n}\right) \sum_{i=1}^{n} \left|\frac{\tilde{y}_i - y_i}{y_i}\right|$, where n is the total number of data points, $\tilde{y}_i$ represents predicted JAR at time step i, and y_i represents actual JAR at time step i.

Baseline WGAN-gp Transformer is evaluated against a state-of-the-art LSTM model, called LoadDynamics (Jayakumar et al. 2020). LoadDynamics employs LSTM model to automatically optimize LSTM for individual workload using Bayesian Optimization. As our baseline, we use the brute force version of LoadDynamics, which performs hyperparamter search for LSTM in predefined hyperparameter search space. For LoadDynamics, we use the same configurations for the model hyperparameters (the number of LSTM layers, the memory cell C size, and the input length n) described in the original version of the paper. For the new cloud workloads (not been evaluated in the original LoadDyanmics paper), we use grid search to find the optimal hyperparameters. The training and testing of LoadDynamics are performed on the same GPU (NVIDIA GeForce RTX 2080 Ti) machine.

Workload	**Load-Dynamics**	**WGAN-gp Transformer**
Facebook-5m	47.20	**42.11**
Facebook-10m	43.68	**39.31**
Alibaba-2018-5m	17.95	**15.76**
Alibaba-2018-10m	16.90	**14.67**
Google-10m	11.49	**10.58**
Google-30m	9.12	**8.34**
Wiki-10m	**1.17**	1.34
Wiki-30m	**1.75**	3.43
Azure-VM-2017-10m	42.63	**41.32**
Azure-VM-2017-30m	29.35	**27.48**
Azure-VM-2017-60m	16.11	**12.77**
Azure-VM-2019-30m	19.74	**15.19**
Azure-VM-2019-60m	13.5	**10.82**
Azure-Func-2019-5m	**1.63**	3.05
Azure-Func-2019-10m	2.06	**1.85**

Table 3: Average prediction errors (MAPE) for cloud workloads

Evaluation Results

Prediction Errors and Inference Overheads We first evaluate the prediction error of WGAN-gp Transformer. Table 3 reports the prediction error of WGAN-gp Transformer and the baseline (LoadDynamics) when predicting 15 workload configurations. While the prediction errors vary with different workload configurations, the results clearly show that WGAN-gp Transformer outperforms the baseline model for most workload configurations. In particular, WGAN-gp Transformer provides up to 5.1% lower prediction errors (MAPE). WGAN-gp Transformer relies on the alternative adversarial training techniques to enforce Lipschitz constraint using gradient penalty with MADGRAD optimizer. Additionally, the alternative adversarial training technique is able to reduce the prediction error against the LSTM's recurrences.

We examine the impact of the model optimizer on the accuracy of WGAN-gp Transformer by comparing the prediction errors of our model with both MADGRAD and Adam optimizer. In this evaluation, Adam optimizer employs the following parameters; $\beta_1 = 0$, $\beta_2 = 0.9$, and learning rate $= 0.0001$ for both generator and critic. As shown in Figure 3, WGAN-gp Transformer with MADGRAD optimizer shows a significant reduction in the prediction errors, indicating that the use of MADGRAD optimizer is the critical factor for more accurate workload prediction.

We also notice that WGAN-gp Transformer can be less accurate than LSTM-based forecasting for three workload datasets, *i.e.,* Wiki-10min, Azure-Func-2019-5m, and Azure-Func-2019-10m. Our further analysis reveals that these workload patterns have relatively stronger seasonality than others. *e.g.,* Wikipedia workloads shown in Figure 1(d). LSTM's memory capability can be better to store more accu-

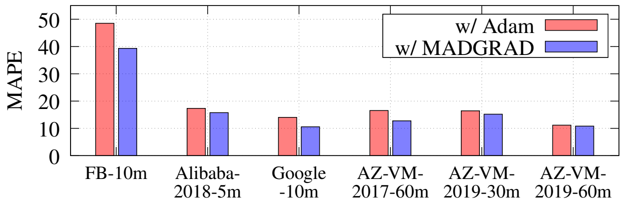

Figure 3: Prediction error comparison between WGAN-gp Transformer with Adam and WGAN-gp Transformer with MADGRAD optimizer. (FB: Facebook, Az: Azure)

rate information for such repeating patterns and yield lower prediction errors.

We also measure the inference overhead (time) of both models when performing the prediction tasks. Figure 4 shows the inference time comparison between WGAN-gp Transformer and LSTM and represents the benefit of WGAN-gp Transformer over the LSTM-based model. The results report that WGAN-gp Transformer has 5× faster inference time over the baseline. On average, the average inference time of WGAN-gp is $4.85ms$, on the other hand, the average inference time of LoadDynamics is $25.57ms$. The faster inference time of WGAN-gp Transformer is because the model processes the input sequence at once, which results in quicker inference time. On the other hand, when LSTM processes the input in the sequence, it processes only one time step at a time, which increases the inference time for processing longer sequences in the prediction tasks.

Auto-Scaling Evaluations on Google Cloud After evaluating the prediction errors and inference overhead, we develop VM auto-scaling systems with WGAN-gp Transformer and LoadDynamics and deploy them on a real-world cloud platform (GCP[5]; Google Cloud Platform) to measure the performance differences in both systems.

The auto-scaling systems consist of a *workload predictor* (WGAN-gp Transformer or LoadDynamics) and a *VM auto-scaler*. The workload predictor determines the predicted job arrivals and provides the prediction results to the auto-scaler. Suppose P_i, predicted at $(i-1)^{th}$ time interval, is the predicted number of jobs arriving at i^{th} time interval to the auto-scaling system and represents the number of VMs to be created in advance. Note that we use an assumption that one job needs an allocation of a single VM. Therefore, with P_i, the auto-scaler creates the P VMs at i^{th} time interval. Assume T_i is the actual number of jobs arriving at i^{th} time interval. If $T_i > P_i$ (the actual job arrivals are greater than the predicted), then it results in *under-provisioning* and, to accommodate extra demand of the jobs, more VMs need to be allocated. In this case, additional time will be needed to finish the jobs due to the VM startup time (**?**). On contrary, if $T_i < P_i$ (actual job arrivals are smaller than the predicted), this results in *over-provisioning* and incurs extra unnecessary cost with the VMs being idle.

Google Cloud's `e2-medium` VMs are used for this evaluation. Facebook and Azure 2019 workloads are evaluated to compare the auto-scaling performance. For the evaluation with the Facebook workload, we use Cloud Suite's *Data Analytics* benchmark in CloudSuite, which performs large amounts of machine learning tasks using MapReduce framework (Ferdman et al. 2012). For the evaluation with Azure 2019 VM workload, we use *In-Memory Analytics* benchmark in CloudSuite, which uses Apache Spark to execute collaborative filtering algorithm in-memory on dataset of user-movie ratings (Ferdman et al. 2012). The evaluation results are reported in Table 4. As shown in the results, the auto-scaling system with WGAN-gp Transformer outperforms the auto-scaling system with the baseline model. With the Facebook workload, the auto-scaler with WGAN-gp Transformer shows a significant reduction in under-provisioning by 27.95%. For the Azure VM 2019 workload, the system with WGAN-gp Transformer has reduced under- and over-provisioning by 2.56% and 1.92%, respectively. The results clearly indicate that the accurate workload prediction from WGAN-gp Transformer can improve the auto-scaling performance running on real-world cloud platforms.

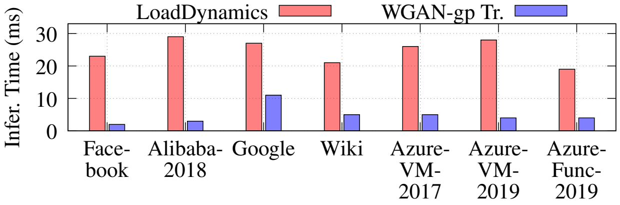

Figure 4: Inference time comparison. (Az: Azure)

Workload	Model	↓ rate (%)	↑ rate (%)
Facebook	LoadDynamics	40.22	10.33
	WGAN-gp Tr.	12.27	16.13
Azure-VM -2019	LoadDynamics	9.63	8.60
	WGAN-gp Tr.	7.07	6.68

Table 4: Auto-scaling evaluation results with VM under-provisioning (↓) rates and over-provisioning (↑) rates

Conclusion

Accurately forecasting user request rates (*e.g.,* job arrival rates) benefits optimizing cloud operating costs and guaranteeing application performance goals through predictive VM auto-scaling. To addresses the problem of job arrival prediction for highly dynamic cloud workloads, we propose a novel time-series forecasting method, called WGAN-gp Transformer, inspired by Transformer network and Wasserstein Generative Adversarial networks. When training with an improved WGAN algorithm, the Transformer network accurately captures dynamic patterns in cloud workloads. We evaluate WGAN-gp Transformer with various real-world cloud workload datasets against the state-of-the-art LSTM model. The results show that our proposed model increases prediction accuracy by up to 5.1%, indicating that the attention mechanism in Transformer network can correctly capture relevant information from past workload sequences to make more accurate predictions. Furthermore, WGAN-gp Transformer significantly reduces inference time by 5× over the state-of-the-art LSTM-based model. Finally,

[5] https://cloud.google.com/

evaluations with real-world cloud deployment show that WGAN-gp Transformer can significantly reduce the under- and over-provisioning rate compared to the baseline model.

Acknowledgments

Jaewoo Lee's work was supported by the National Science Foundation under Grant No. CNS-1943046.

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein Generative Adversarial Networks. In *International Conference on Machine Learning*.

Calheiros, R. N.; Masoumi, E.; Ranjan, R.; and Buyya, R. 2015. Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS. *IEEE Trans. on Cloud Computing*, 3(4).

Chen, Y.; Ganapathi, A.; Griffith, R.; and Katz, R. 2011. The Case for Evaluating MapReduce Performance Using Workload Suites. In *IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems*.

Cortez, E.; Bonde, A.; Muzio, A.; Russinovich, M.; Fontoura, M.; and Bianchini, R. 2017. Resource Central: Understanding and PredictingWorkloads for Improved Resource Management in Large Cloud Platforms. In *ACM Symp. on Operating Systems Principles*.

Defazio, A.; and Jelassi, S. 2021. Adaptivity without Compromise: A Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization. *arXiv preprint arXiv:2101.11075*.

Ferdman, M.; Adileh, A.; Kocberber, O.; Volos, S.; Alisafaee, M.; Jevdjic, D.; Kaynak, C.; Popescu, A. D.; Ailamaki, A.; and Falsafi, B. 2012. Clearing the Clouds: A Study of Emerging Scale-out Workloads on Modern Hardware. *ACM Sigplan Notices*, 47(4).

Goodfellow, I. J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative Adversarial Networks. *arXiv preprint arXiv:1406.2661*.

Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; and Courville, A. 2017. Improved Training of Wasserstein GANs. *arXiv preprint arXiv:1704.00028*.

Hao, J.; Jiang, T.; Wang, W.; and Kim, I. K. 2021a. An Empirical Analysis of VM Startup Times in Public IaaS Clouds. In *IEEE International Conference on Cloud Computing*.

Hao, J.; Jiang, T.; Wang, W.; and Kim, I. K. 2021b. An Empirical Analysis of VM Startup Times in Public IaaS Clouds: An Extended Report. *CoRR*, abs/2107.03467.

Hochreiter, S.; and Schmidhuber, J. 1997. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780.

Islam, S.; Venugopal, S.; and Liu, A. 2015. Evaluating the Impact of Fine-scale Burstiness on Cloud Elasticity. In *ACM Symposium on Cloud Computing*.

Jayakumar, V. K.; Lee, J.; Kim, I. K.; and Wang, W. 2020. A Self-Optimized Generic Workload Prediction Framework for Cloud Computing. In *IEEE International Parallel and Distributed Processing Symposium*.

Jiang, Y.; Chang, S.; and Wang, Z. 2021. TransGAN: Two Pure Transformers Can Make One Strong GAN, and That Can Scale Up. *arXiv preprint arXiv:2102.07074*.

Jiang, Y.; Perng, C.-S.; Li, T.; and Chang, R. N. 2011. ASAP: A Self-Adaptive Prediction System for Instant Cloud Resource Demand Provisioning. In *IEEE International Conf. on Data Mining*.

Kim, I. K.; Wang, W.; Qi, Y.; and Humphrey, M. 2016. Empirical Evaluation of Workload Forecasting Techniques for Predictive Cloud Resource Scaling. In *IEEE International Conference on Cloud Computing*.

Kim, I. K.; Wang, W.; Qi, Y.; and Humphrey, M. 2018. Cloudinsight: Utilizing a council of experts to predict future cloud application workloads. In *IEEE International Conference on Cloud Computing*.

Kim, I. K.; Wang, W.; Qi, Y.; and Humphrey, M. 2020. Forecasting Cloud Application Workloads with CloudInsight for Predictive Resource Management. *IEEE Transactions on Cloud Computing*, 1–1.

Kumar, S.; Muthiyan, N.; Gupta, S.; A.D., D.; and Nigam, A. 2018. Association Learning based Hybrid Model for Cloud Workload Prediction. In *International Joint Conference on Neural Networks*.

Lim, B.; and Zohren, S. 2021. Time-series Forecasting with Deep Learning: A Survey. *Philosophical Transactions of the Royal Society A*, 379(2194): 20200209.

Lin, H.; Qi, X.; Yang, S.; and Midkiff, S. 2015. Workload-driven VM Consolidation in Cloud Data Centers. In *IEEE International Parallel and Distributed Processing Symposium*.

Loff, J.; and Garcia, J. 2014. Vadara: Predictive Elasticity for Cloud Applications. In *IEEE International Conference on Cloud Computing Technology and Science*.

Mao, M.; and Humphrey, M. 2011. Auto-scaling to Minimize Cost and Meet Application Deadlines in Cloud Workflows. In *International Conference on High Performance Computing Networking, Storage and Analysis*.

Mao, M.; and Humphrey, M. 2012. A Performance Study on the VM Startup Time in the Cloud. In *IEEE International Conference on Cloud Computing*.

Nguyen, C.; Klein, C.; and Elmroth, E. 2019. Multivariate LSTM-Based Location-Aware Workload Prediction for Edge Data Centers. In *Int'l Symposium on Cluster, Cloud and Grid Computing*.

Roy, N.; Dubey, A.; and Gokhale, A. 2011. Efficient Autoscaling in the Cloud Using Predictive Models for Workload Forecasting. In *IEEE International Conference on Cloud Computing*.

Rzadca, K.; Findeisen, P.; Swiderski, J.; Zych, P.; Broniek, P.; Kusmierek, J.; Nowak, P.; Strack, B.; Witusowski, P.; Hand, S.; and Wilkes, J. 2020. Autopilot: workload autoscaling at Google. In *ACM European Conference on Computer Systems*.

Shen, Z.; Subbiah, S.; Gu, X.; and Wilkes, J. 2011. Cloudscale: Elastic Resource Scaling for Multi-tenant Cloud Systems. In *ACM Symposium on Cloud Computing*.

Tsai, Y.-H. H.; Bai, S.; Yamada, M.; Morency, L.-P.; and Salakhutdinov, R. 2019. Transformer Dissection: A Unified Understanding of Transformer's Attention via the Lens of Kernel. *arXiv preprint arXiv:1908.11775*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention is All You Need. *arXiv preprint arXiv:1706.03762*.

Wilkes, J. 2011. More Google cluster data. http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html. Accessed: 2021-09-01.

Wood, T.; Cherkasova, L.; Ozonat, K.; and Shenoy, P. 2008. Profiling and Modeling Resource Usage of Virtualized Applications. In *ACM/IFIP/USENIX International Middleware Conference*.

Wu, S.; Xiao, X.; Ding, Q.; Zhao, P.; Wei, Y.; and Huang, J. 2020. Adversarial Sparse Transformer for Time Series Forecasting. In *Advances in Neural Information Processing Systems*.

Xu, B.; Wang, N.; Chen, T.; and Li, M. 2015. Empirical Evaluation of Rectified Activations in Convolutional network. *arXiv preprint arXiv:1505.00853*.

Yadwadkar, N. J.; Ananthanarayanan, G.; and Katz, R. 2014. Wrangler: Predictable and Faster Jobs using Fewer Resources. In *ACM Symposium on Cloud Computing*.