# Wasserstein Adversarial Transformer for Cloud Workload Prediction

by

Shivani Gajanan Arbat

(Under the Direction of In Kee Kim)

## Abstract

Understanding job arrival rates is critical for predicting future workloads to provision cloud resources for reducing cloud costs and meeting the applications' performance goals. However, developing an accurate model to forecast job arrival rates is non-trivial due to the dynamic nature of workloads. The current state-of-the-art LSTM model uses recurrences for predictions, causing increased complexity and degraded computational efficiency. Therefore, to address this problem, 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 show WGAN-gp Transformer achieves 5× faster inference time with up to 5.1% higher prediction accuracy than the state-of-the-art technique. We then apply our model to auto-scaling mechanisms on cloud platforms and show that the WGAN-gp-Transformer-based auto-scaling mechanism outperforms LSTM-based auto-scaling by reducing VM over-provisioning and under-provisioning by at least 1.92% and 2.56% respectively.

Index words: Cloud Workload prediction, Wasserstein Generative Adversarial Networks, Time Series Forecasting

# Wasserstein Adversarial Transformer for Cloud Workload Prediction

by

Shivani Gajanan Arbat

B.E., University of Pune, India 2014

A Thesis Submitted to the Graduate Faculty of the

University of Georgia in Partial Fulfillment of the Requirements for the Degree.

Master of Science

Athens, Georgia

2021

# Wasserstein Adversarial Transformer for Cloud Workload Prediction

by

Shivani Gajanan Arbat

|  |  |
|---|---|
| Major Professor: | In Kee Kim |
| Committee: | Jaewoo Lee |
|  | Lakshmish Ramaswamy |

*Dedicated to my Mother, Father, Akka, Devi, Rushu, and Devashish*

*and to my dearest friends Neerja, Shikha, Teju, and Kiran*

# Acknowledgments

I appreciate each of the passionate and inspiring people whom I met during my graduate school journey. First of all, I wish to thank my advisor, Dr. In Kee Kim, for his guidance, direction, and constant motivation in completing this work. I am thankful for his patience and constructive edits as I worked to meet his high research standards. It is my great pleasure to thank my committee members: I am thankful to Dr. Jaewoo Lee for his guidance and supervision of this work. Dr. Lee guided and challenged me and spent a generous amount of time with me at every required step of completing this work. I would like to thank Dr. Lakshmish Ramaswamy for his invaluable insights. His courses helped me built a foundational understanding demanded by this work.

I also want to express my gratitude to Dr. Brian T. Forschler for his support and encouragement at every step of my academic growth. I sincerely appreciate the valuable experiences he exposed me to and have great faith in me throughout the years I worked as a graduate research assistant. I also want to thank my colleague at Cloud Lab - Jianwei Hao, who helped me with the machine setup for experiments.

I am grateful to all the friends I made in Athens who helped me through tough times and encouraged me to finish the work I started. I appreciate the friends I met at UGA: Shrutika, Ananta, Sushruth, Allison, and so many others whose names would fill an entire page. The camaraderie and great times spent together in beautiful Athens are something I will cherish and will miss.

Lastly, my gratitude goes to my family and friends to whom this work is dedicated. Their unconditional love, care, support, and sacrifices made this possible. I owe a most heartfelt debt of gratitude to Ranju Akka. It is impossible to measure all that you have done to get started with this journey, but I am grateful for all the big and small things you have done for me.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

Resource provisioning and auto-scaling of cloud resources are essential operations to optimize cloud costs and the performance of cloud applications [1–4]. Auto-scaling dynamically performs scale-out and scale-in operations as application workloads fluctuate. e.g., change in user requests to cloud applications. The scale-out operation increases the number of VMs (Virtual Machines) for the cloud application as the workload increases, and the cloud application leverages enough amount of VMs and satisfies its performance goal. i.e., SLOs (Service Level Objectives), QoS (Quality of Services), job deadline, and applications' response time. On the other hand, the scale-in operations automatically downsize the number of provisioned VMs by terminating idle VMs when the workload decreases and help the cloud application to minimize the cloud cost. While auto-scaling offers benefits to cloud applications, the typical limitations of auto-scaling are delays in resource provisioning, e.g., VM startup delay and termination delay, due to its *reactive* nature [5, 6], hence offering suboptimal cloud resource management.

To address the reactive nature of auto-scaling, predictive auto-scaling approaches have been deeply investigated [6–19]. Predictive auto-scaling mechanisms commonly have two components, which are *workload predictor* and *auto-scaling module*. In particular, the workload predictor is used to forecasts the future workload changes to cloud applications, and an essential step for designing the workload predictor is to understand job arrival rates[1] to cloud applications. For designing workload predictors, diverse

[1]In this thesis, workloads, user requests, and job arrival rates are interchangeably used.

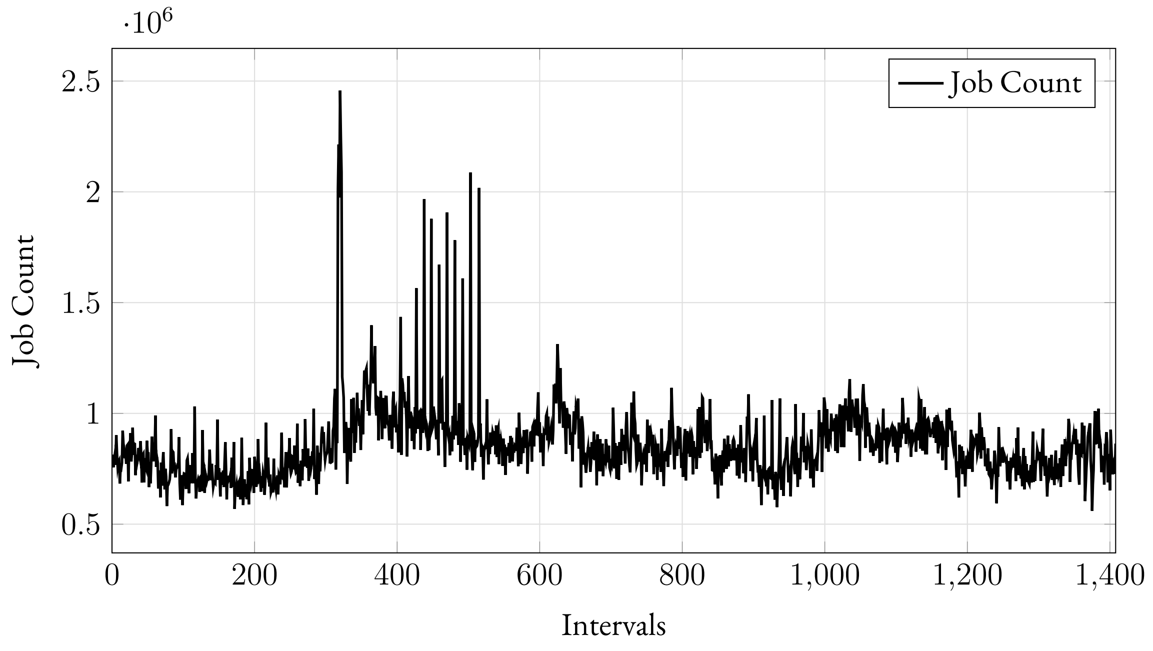


Figure 1.1: Google Cluster Trace (30 minutes interval) [23]

mechanisms have been leveraged, including statistical time-series methods [7, 8, 12], traditional machine learning [6, 9, 13], ensemble learning [10, 11, 14, 18], and deep learning [15–17, 19]. The state-of-the-art approach to predict cloud workloads is to employ a combination of LSTM (Long Short-Term Memory) and Bayesian optimization [19], specifically leveraging the power of LSTM to understand job arrival rate information for longer periods of time using recurrences [20]. However, these recurrences resulted in increasing complexity and poor computational efficiency over time as input sequences grows 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 [21] as shown in the trace of Google Cluster Data [22] (Figure 1.1). These sudden spikes in Google Cluster trace show unprecedented changes in cloud workloads over time. Therefore, there is an urgent need to develop a novel cloud workload predictor with high accuracy and low computational overhead for predictive auto-scaling.

This thesis presents a novel time-series forecasting model for cloud resource provisioning, called WGAN-gp (Wasserstein Generative Adversarial Network with gradient penalty) Transformer. The proposed WGAN-gp Transformer is inspired by Transformer networks [24] and improved WGAN (Wasserstein Generative Adversarial Networks) [25]. Our proposed method adopts a Transformer network as the generator and a multi-layer perceptron network as a critic to improve the overall forecasting performance. WGAN-gp Transformer also employs MADGRAD (Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization) as the model optimizer for both, generator and critic. MADGRAD is designed using dual averaging formulation of AdaGrad [26], adapted for deep learning settings. The motivation to use MADGRAD is to achieve better convergence compared to widely adopted Adam optimizer. Also it is significantly faster at earlier iterations and also achieves lower prediction error compared to Adam [27].

The comprehensive evaluation with representative cloud workload datasets shows that WGAN-gp Transformer consistently performs better, yielding lower prediction errors. The performance of WGAN-gp Transformer is compared against state-of-the-art LoadDynamics model [19]. The evaluation results with 15 workload configurations from seven representative cloud datasets show that WGAN-gp Transformer achieves up to 5.1% lower prediction error and 5× faster prediction time compared to the state-of-the-art LoadDyanmics model for dynamic cloud workload prediction. The auto-scaling evaluation are performed with Azure 2019 and Facebook dataset. Evaluation results show that WGAN-gp Transformer performs significantly better to reduce the under-provisioning for Facebook dataset by 27.95%. For Azure 2019 dataset, the over-provisioning and under-provisioning is reduced by 1.92% and 2.56% respectively.

As a result, this thesis has the following research contributions:

1. Proposed a novel forecasting model for diverse cloud workload, inspired by Wasserstein Generative Adversarial Network improving transformer networks for time series prediction.

2. Improved accuracy compared to state of the art model for predicting cloud workloads. The results are evaluated by comparing test errors with other prediction models.

3. Reduced on average inference time by 5 times compared to state of the art.

4. Improvement in auto-scaling performance by reduced over-provisioning and under-provisioning.

The rest of this thesis is organized as follows: Chapter 2 discusses existing approaches to predicting cloud workloads and highlights adversarial methods for time series forecasting in previous work. Chapter 3 describes the preliminaries and background of our approach design. Chapter 4 discusses Wasserstein GANs, their advanced setup and discusses WGAN-gp Transformer in detail. Chapter 5 describes the performance evaluation of WGAN-gp Transformer with seven representative cloud workload datasets. Finally, Chapter 6 concludes this thesis by summarizing our analysis, research contribution, and vision of future work.

# Chapter 2

# Related Work

The critical challenge of effective auto-scaling in the cloud can be addressed job arrival rate or user rate level. Many studies have analyzed user request rates in the form of time series data to make a prediction. [1, 2, 18]. Various time series models like Weighted Moving Average [4, 28–31], Autoregressive models (AR) [32–34] and its variations (ARMA, ARIMA) [8, 35–37] have been applied to cloud workload forecasting. Statistical approaches were able to model time series, which show a cyclic trend or seasonal pattern. However, a single statistical model-based approaches appeared to conduct sub-optimal predictions for cloud workloads as they are continuously and dynamically changing over time. Also, same model proved effective on one type of workload does not guarantee to perform optimally on other unknown workload patterns [6]. Besides, Machine Learning (ML) models, e.g., Linear Regression [2, 21, 38, 39], Support Vector Machine [9], Random Forest [13], and Gradient Boosting [13], were also employed to improve forecasting quality achieved using statistical models.

For cloud workloads, no single best predictor based on statistical models or ML models proved effective for most cloud workloads. Multi-predictor based approaches subdue these limitations [10, 11, 14, 40–43]. Especially, Kim et al. [14] proposed CloudInsight, a machine learning paradigm called ensemble learning, which selects the best predictor to forecast user requests in cloud workloads from a group of ML models - Random Forest, ARIMA, and Support Vector Machine. The choice of models in the ensemble settings is subjective and can be changed to other combinations of statistical or ML models. CloudInsight significantly improved forecasting accuracy over using one single predictor approach. However, a drawback of this approach is to train multiple predictors in parallel to select top predictors, which presents needless overhead in computation and increases inference time.

The applications of neural networks in time series forecasting provide improved accuracy across multiple domains. Neural networks learn to encode relevant historical information from time series data into intermediate feature representation to make a final forecast with series of non-linear neural network layers [44]. Various encoder architectures of neural networks learn to capture this relevant temporal information using recurrences or convolutions when adapted to time series datasets. For cloud workloads, LSTM [20] and its variations are studied to forecast the resource demands or user requests. [15–17, 19, 45–47]. Jayakumar et al. [19] proposed LoadDynamics, a self-optimized generic workload prediction framework for cloud workloads. In LoadDynamics, the self-optimization is performed using Bayesian Optimization, which readily optimizes LSTM hyperparameters for different types of workloads. LoadDynamics conducted comprehensive evaluations with different cloud workloads, from Google, Azure (2018), and web applications, to forecast job arrival rates. The brute-force approach of LoadDynamics is considered as the baseline in this work. 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. Also, LSTMs are prone to overfitting, which leads to poor performance of the model.

An efficient alternative and improvement on recurrences or convolutions is the introduction of attention mechanism in Transformer networks, adapted from NLP literature [24]. For time series data, attention layers in the Transformer network allow the model to focus on temporal information in a parallel manner anywhere in the input sequence [44]. Neo Wu et al. [48] employed a Transformer network to forecast time series data. Variation of Transformer network are also applied to problem of time series forecasting by introducing sparse self-attention, convolutional self-attention [49, 50]. To improve over the performance of vanilla version of Transformer, Sifan Wu et al. [50] proposed Adversarial Sparse Transformer (AST), based on GANs. AST uses sparse attention mechanism to improve the prediction performance at sequence level by employing sparse transformer as generator and multi-layer perceptron as discriminator. Even though AST significantly improved forecasting performance for time series, however, for dynamic cloud workloads, AST has limitations for losing information in long-term forecasting due to the sparse point-wise connections with assumed difficulty to train GANs.

On the other hand, to effectively predict job arrival rates by capturing long term information from the time series data, our method, WGAN-gp Transformer, proposes to train the Transformer network using improved WGAN-gp algorithm [51]. In chapter 5, the evaluations are performed on 15 workload configurations of cloud workloads from Google, Azure, Facebook, Alibaba, and Wikipedia. The evaluations results show that WGAN-gp Transformer outperforms state-of-the-art workload predictor.

The next chapter introduces the basic concepts to build a foundation understanding of the Wasserstein Generative Adversarial networks [25].

# Chapter 3

# Background

This chapters provides some background on time series, general explanation of components of our proposed time series forecasting model designed for cloud workloads. We begin by defining different elementary distances and divergences between two probability distributions, $\mathbb{P}_r$ and $\mathbb{P}_g$ in $\mathcal{X}$ , a compact metric space. We discuss GANs and their limitations. We later introduce WGANs and elaborate how they improve the stability of learning where GANs have proved to fall short due their discussed shortcomings. In our proposed approach, Transformer Network introduced in [24] is used *generator* and a multi-layer perceptron network as *critic*.

## Time series

A univariate time series is defined as a sequence of measurements for a same variable collected over time. Let $\{\mathbf{x}_{i,1:t_n}\}_{i=1}^{S}$ is univariate time series $S$, where $\mathbf{x}_{i,1:t_n} = [x_{i,1}, x_{i,2}, ..., x_{i,t_n}]$ and $x_{i,t} \in \mathbb{R}$ is the value of time series $i$ at time $t$. In this work, we study a univariate time series data of job arrival rates (JARs) or user requests rate collected at regular time intervals from variety of cloud workloads.

## Probability Distance Metrics

Probability Distance Metrics are classified into two classes - *F-divergences* (F-Div) and *Integral Probability Metrics* (IPM) [52]. F-divergences calculates distance using division operation of probability distribution ($\mathbb{P}_r$ / $\mathbb{P}_g$). IPM determines distance using calculating difference between probabilities ($\mathbb{P}_r - \mathbb{P}_g$). *Kullback-Leibler* (KL) divergence and*Jensen-Shannon* (JS) divergence probability distance metrics belong to F-Div class. *Earth-Mover* (EM) distance or *Wasserstein*-1 distance belong to IPM class. There are other probability distance metrics, however we will discuss these three probability distance metrics in this chapter. Let $\mathbb{P}_r$ and $\mathbb{P}_g$ be two distributions in $\mathcal{X}$, a compact metric space. We define elementary distances and divergences as follows:

The *Kullback-Leibler* (KL) divergence is defined as :

$$KL(\mathbb{P}_r||\mathbb{P}_g) = \int log\left(\frac{\mathbb{P}_r(x)}{\mathbb{P}_g(x)}\right)\mathbb{P}_r(x)dx \tag{3.1}$$

The*Jensen-Shannon* (JS) divergence probability distance metric is defined as :

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r||\mathbb{P}_m) + KL(\mathbb{P}_g||\mathbb{P}_m) \tag{3.2}$$

where, $\mathbb{P}_m$ is the average of two probability distribution $\mathbb{P}_r$ and $\mathbb{P}_g$.

The *Earth-Mover* (EM) distance or *Wasserstein*-1 distance is informally termed as the minimum cost of transporting mass to transform distribution $\mathbb{P}_g$ to $\mathbb{P}_r$ [51]. EM or *Wasserstein*-1 distance is defined as :

$$W(\mathbb{P}_r, \mathbb{P}_g) = \underset{\gamma \in \prod(\mathbb{P}_r, \mathbb{P}_g)}{inf} \mathbb{E}_{(x,y)\ \gamma}\left[||x-y||\right] \tag{3.3}$$

where, $\prod(\mathbb{P}_r, \mathbb{P}_g)$ is set of all joint distributions $\gamma(x,y)$ whose marginals are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. $\gamma(x,y)$ indicates "mass" needed to transport from $x$ to $y$ in order to transform probability distribution $\mathbb{P}_r$ to $\mathbb{P}_g$. The *Wasserstein*-1 distance is optimal "cost" of this transport operation.

The use of these different classes of the probability distance metrics addresses the variations of GAN algorithms [25].

### Generative Adversarial Networks

Generative Adversarial Networks [53] are *adversarial nets* framework, which trains two models simultaneously - a *generative* model $G$ and a *discriminative* model $D$. This training strategy is a min-max game between two competing networks. The model $G$ implicitly defines a generator probability distribution $\mathbb{P}_g$ as the distribution of samples $G(z)$ obtained when $z \sim p_z$ (the input z to the generator is sampled from some simple noise distribution - a uniform distribution or spherical Gaussian distribution), on the other hand, the model $D$ is trained to estimate the probability of identifying that from which distribution the sample belong. $D(x)$ in equation 3.4 represents the discriminator probability that $x$ belongs to data rather than $\mathbb{P}_g$. In this adversarial modelling framework, model $G$ and model $D$ participates in *min-max* game. Thus the value function $V(D, G)$ is defined as :

$$\underset{G}{min}\, \underset{D}{max}\, V(D,G) = \mathbb{E}_{x\sim\mathbb{P}_r}[logD(x)] + \mathbb{E}_{z\sim\mathbb{P}_g}[log(1 - D(\tilde{x}))] \tag{3.4}$$

where $\mathbb{P}_r$ is the data distribution and $\mathbb{P}_g$ is the model distribution implicitly defined by $\tilde{x} = G(z)$.

The minimization of value function 3.4 is achieved by minimizing the *Jensen-Shannon* divergence (equation 3.2) between $\mathbb{P}_r$ and $\mathbb{P}_g$ [53]. However, this leads to vanishing gradient problem as the discriminator saturates. Even though Goodfellow et al. [53] discussed about maximizing $\mathbb{E}_{z\sim\mathbb{P}_g}[log(D(\tilde{x})]$ by using updated version of loss function to mitigate the vanishing gradient problem, a good discriminator can still aggravate this issue [54].

### Wasserstein GAN

An alternative training method of GAN was proposed in Arjovsky et al. [25]. The divergences minimized in a typical GAN are not continuous to the generator's parameter. Thus a different probability distance metric *Wasserstein*-1 distance was proposed in Wasserstein GANs. The discriminator is referred as *critic* in the literature. Since the *critic* is not trained to classify it does not do much on its own. The EM or *Wasserstein*-1 distance is continuous and differentiable which implies that the critic can be trained optimally. In WGAN, the role of critic is to learn to update the weights to find function that maximizes the Wasserstein distance between between $\mathbb{P}_r$ and $\mathbb{P}_g$. This eliminates the problem with using JS distance, as the JS saturates locally showing vanishing gradients.

The WGAN value function using Kantorovich-Rubinstein [55] duality can be defined as :

$$\underset{G}{min}\ \underset{D \in \mathcal{D}}{max}\ \mathbb{E}_{x \sim \mathbb{P}_r}[D(x)] - \mathbb{E}_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] \tag{3.5}$$

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

Equation 3.5 is minimized with respect to generator parameters $\theta_g$ which minimizes $W(\mathbb{P}_r, \mathbb{P}_g)$. This results in generator which is better optimized compared to its GAN counterpart [25]. Empirical evaluations in Arjovsky et al. [25] show that WGANs have proved to be more robust than GANs. Also WGANs are able to learn the distribution without the problem of mode collapse which is evident in case of GANs. In order to enforce the Lipschitz constraint on *critic*, weight clipping is performed within a compact space of $[-c, c]$. The set of Lipschitz functions are subset of $k$-Lipschitz functions for some $k$ which depends on $c$ and the *critic* architecture. Weight clipping is introduced to enforce Lipschitz constraint will not allow the *critic* to saturate and force to converge in linear function [25]. WGAN optimization process should be balanced to avoid vanishing gradients or exploding gradients problem, with careful tuning of the clipping threshold $c$.

However, Gulrajani et al. [51] discussed the drawbacks of using weight clipping to enforce Lipschitz constraint to address training challenges for *critic* and evident vanishing gradient problem on selecting improper value of clipping parameter $c$. In the next chapter, we discuss improved training of WGANs with gradient penalty eliminating need of weight clipping [51].

# Chapter 4

# Wasserstein Adversarial (WGAN-gp) Transformer

As discussed in the Introduction, cloud workloads have a collectively cyclic and bursty combination of job arrival rate, subject to randomness or change over time. For such cloud workloads, this work proposes a novel model for time series forecasting - WGAN-gp Transformer. The proposed model is designed on the foundation of Transformer and Wasserstein Generative Adversarial Network. In this chapter, we explain Wasserstein GAN with gradient penalty (WGAN-gp), an improved training algorithm for Wasserstein GANs proposed in Gulrajani et al. [51]. In WGAN-gp Transformer, the generator is Transformer Network, and critic is a three-layer multi-layer perceptron. The attention mechanism in the Transformer network is crucial for learning long-term dependencies in the cloud workload time-series data. We explain the attention mechanism and encoder-decoder architecture of the Transformer network, which builds the core of the WGAN-gp Transformer's *generator*. We briefly discuss the recently proposed new optimizer; a Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization (MADGRAD) [27] method to perform hyperparameter optimization before we introduce our proposed model - Wasserstein Adversarial Transformer (WGAN-gp Transformer).

## 4.1 Wasserstein GAN-gp

While GANs (discussed in section 3) are robust generative networks, GANs can suffer from training instabilities [54]. This issue can be addressed by introducing WGANs, which provide an alternate method of GAN training using weight clipping. However, the weight clipping $c$ is crucial hyperparameter which significantly affects the WGAN training. Irregular value surfaces are generated due to hard clipping of the magnitude of each weight. Empirical evaluations shown in Gulrajani et al. [51] using other weight constraints like L2 norm clipping, weight normalization leads to similar problems. Soft constraints like L1 and L2 weight decay also lead to the same problem. Thus an alternative way to enforce the Lipschitz constraint was proposed in Gulrajani et al. [51]. Instead of weight clipping (hard weight clipping or soft weight clipping), Gulrajani et al. [51] proposed to penalize gradients for both real and generated data by improving the objective function equation 3.5 with an additional gradient penalty term.

### 4.1.1 Gradient Penalty

A differentiable function $f^*$ is 1-Lipschitz if and only if it has gradient norm of 1 almost everywhere. In addition to the original *critic* loss, gradient penalty term is added (equation 3.5) for a fixed gradient penalty coefficient $\lambda$. The gradient penalty coefficient $\lambda$ affects the magnitude of penalty term. The value of the coefficient is set to $\lambda = 10$, which is default value suggested in WGAN-gp algorithm. Thus, the gradient norm of the critic's output is directly constrained to be 1 for its respective input [51]. Thus the new objective is :

$$L = \underbrace{\mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x\sim\mathbb{P}_r}[D(x)]}_{\text{Original critic loss}} + \underbrace{\lambda\,\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(||\nabla_{\hat{x}}D_w(\hat{x})||_2 - 1)^2]}_{\text{Gradient penalty}} \tag{4.1}$$

We have used a one-sided penalty in our proposed approach. That is, the gradient norm is encouraged to stay below 1. The alternative technique to enforce the Lipschitz constraint has proven to improve training speed and quality. Furthermore, as discussed in Gulrajani et al. [51], adding the penalty term invalidates the need to have batch normalization since the norm of the critic's gradient is penalized concerning each input independently. Instead, layer normalization is recommended as a replacement for batch normalization.

### 4.1.2 Improved training of WGANs with gradient penalty (WGAN-gp)

For training of Wasserstein GAN with gradient penalty (WGAN-gp), default values are set for $n_{critic} = 5$, $\lambda = 10$. To train both *generator* and *critic* Adam optimizer parameters are set as $\beta_1 = 0$ and $\beta_2 = 0.9$ with learning rate $\alpha = 0.0001$. $n_{critic} = 5$ value is set as default suggested in WGAN-gp algorithm [51], which implies there will be five iterations of the *critic* per *generator* iteration. Below is the algorithm for WGAN-gp :

**Algorithm 1:** Wasserstein GAN with gradient penalty proposed in Gulrajani et al. [51]

```
Require: $\lambda$, the gradient penalty coefficient. m, the batch size.
         $n_{critic}$, the number of iterations of the critic per
         generator iteration.
Require: initial critic parameters $w_0$, initial generator
         parameters $\theta_0$.
while $\theta$ has not converged do
    for $t = 1, ...., n_{critic}$ do
        for $i = 1, ...., m$ do
            Sample real data $x \sim \mathbb{P}_r$, a latent variable $z \sim p(z)$, a
             random number $\epsilon \sim U[0, 1]$.
            $\tilde{x} \leftarrow G_\theta(z)$
            $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\tilde{x}$
            $L^{(i)} \leftarrow D_w(\tilde{x}) - D_w(x) + \lambda(||\nabla_{\hat{x}} D_w(\hat{x})||_2 - 1)^2$
        end
        $w \leftarrow$ Adam($\nabla_w \frac{1}{m} \sum_{i=1}^{m} L^{(i)}, w, \alpha, \beta_1, \beta_2$)
    end
    Sample a batch of latent variables $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$
    $\theta \leftarrow$ Adam($\nabla_\theta \frac{1}{m} \sum_{i=1}^{m} -D_w(G_\theta(z)), \theta, \alpha, \beta_1, \beta_2$)
end
```

## 4.2 Transformer Network

In this section, we discuss the vanilla version of Transformer proposed in Vaswani et al. [24], which is type of encoder-decoder architecture. Transformer network architecture is based purely on attention mechanisms, eliminating the need for recurrences or convolutions. Unlike recurrences [56] or convolution [57], Transformer processes the entire sequence at the same time. Before any processing is done, the input sequences are translated into high dimensional vector of dimension $d_{model}$.

### 4.2.1 Attention Mechanism

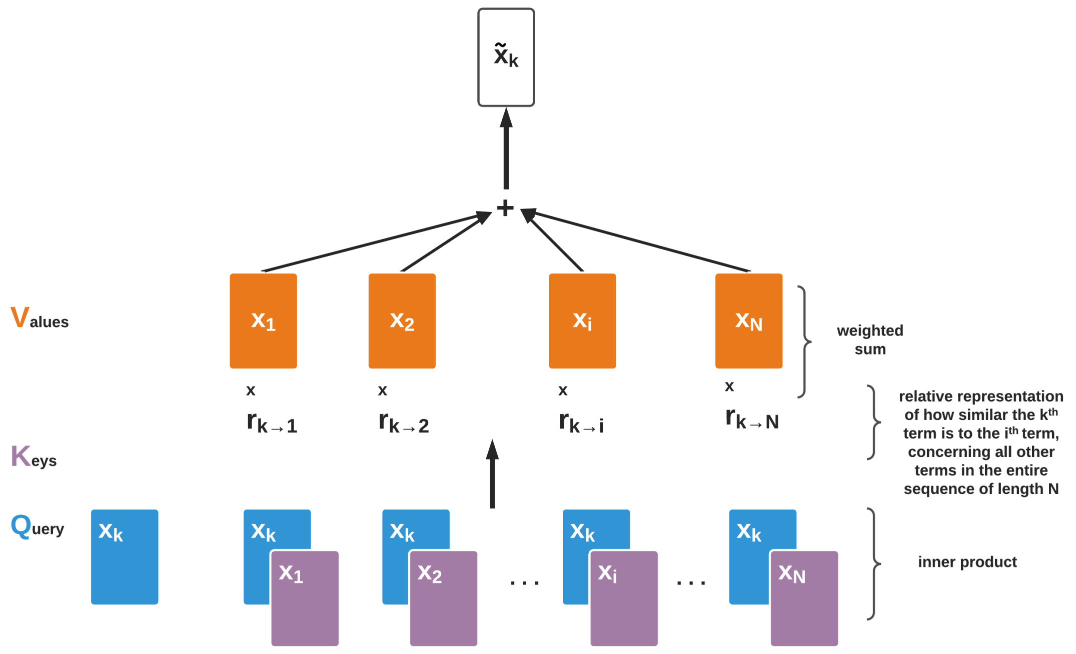
Attention mechanism in the Encoder-decoder architecture of Transformer facilitates capturing long-term dependencies in time-series in tasks like time-series forecasting. Attentions mechanism can be described as quantifying the similarity of entire sequence of length $N$ to $k_{th}$ term. The attention function allows to map a *query* and a set of *key-value* pairs [1] to compute an output (illustrated in Figure 4.1). The output is computed by taking a weighted sum of *values*; these weights represent relative representation of how similar the $k_{th}$ term is to $i_{th}$ term, with respect to all of the other $N$ terms in the sequence.

Figure 4.1: Sequence of length $N$ representing $i_{th}$ and $k_{th}$ term

Figure 4.2 illustrates the attention mechanism. First we take inner product between *query* $x_k$ and entire sequence $\mathbf{x}_{1,2,\ldots,i,\ldots,N}$ and $N \in \mathbb{R}$ is length of sequence [2]. The inner products can be negative or positive. Thus, exponentiate these inner products to make them positive. This gives us the relative representation $r_{k\to 1,2,..,i,..,N}$ which corresponds to similarity between terms. To get the attention score $\tilde{x}_k$, the weighted sum of *values*, with $r_{k\to 1,2,..,i,..,N}$ corresponding to weighted sum.

[1] Here *query, key, values*, and output are all vectors

[2] Here time $t$ is not considered for ease of understanding. Thus, here sequence can be assumed as sequence of numbers.



$\tilde{x}_k$
+
Values
Keys
Query
$x_1$
$x_2$
$x_i$
$x_N$
x
$r_{k\to1}$
$r_{k\to2}$
$r_{k\to i}$
$r_{k\to N}$
weighted sum
relative representation of how similar the kth term is to the ith term, concerning all other terms in the entire sequence of length N
$x_k$
inner product

Figure 4.2: Attention Mechanism

### Scaled Dot-Product Attention

In Vaswani et al. [24], the attention function on a set of queries is computed simultaneously, represented as matrix $Q$. Similar to matrix $Q$, keys and values are represented as matrix $K$ and matrix $V$, respectively. Dot-product attention is comparatively faster and space-efficient than Additive attention discussed in Vaswani et al. [24]. In this thesis, we have used a scaled dot-product attention mechanism. Thus, the attention function on matrices $Q$, $V$, and $K$ is computed as :

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{4.2}$$

where $d_k$ is dimension of queries and keys.

### Multi-head attention

Multi-head attention can be defined as a function computing several single attention functions on linearly projected *query, key,* and *value*[3] $h$ times with different, learned projections of $d_k$, $d_k$ and $d_v$. $d_k$ represents dimensions of *query* and *key*. $d_v$ represents dimension of *value*. Individual attention functions are computed in parallel, yielding $d_v$ dimension output values. These output values are then concatenated to resulting final output value as depicted in Figure 4.3. Multi-head attention function [24] can be written as :

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{4.3}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{4.4}$$

where the projections are parameter matrices $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W_i^O \in \mathbb{R}^{hd_v \times d_{model}}$.

In this thesis, we have employed $h$ and $d_{model}$ as hyperparameters[4] which we discuss in later Chapters.

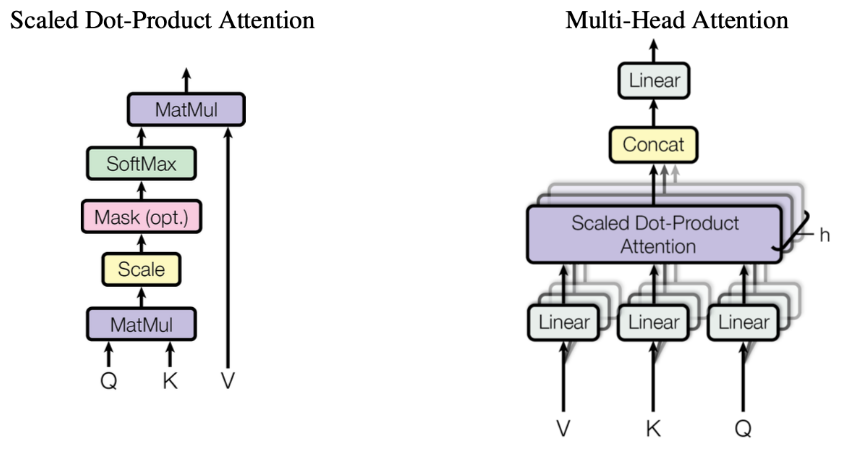


Figure 4.3: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention [24]

[3]*query, key,* and *value* are of $d_{model}$ dimensions. The resulting *query, key,* and *value* matrices will be of dimension $d_{model} \times d_k$, $d_{model} \times d_k$ and $d_{model} \times d_v$

[4]$h$ notation in this work is also refereed as $nhead$ which represents multiple attention heads.

### 4.2.2 Positional Encoding

Temporal (positional) information needs to be explicitly embedded in input sequences since the Transformer does not have implicit mechanisms like recurrences or convolutions; Transformer networks are order-agnostic [24][58]. Positional embedding is mixture of sine or cosine functions. Similar to Vaswani et al. [24], this work implements sine and cosine functions of different frequencies represented in below equations :

$$PE_{(pos,2i)} = sin\left(\frac{pos}{10000^{\left(\frac{2i}{d_{model}}\right)}}\right) \tag{4.5}$$

$$PE_{(pos,2i+1)} = cos\left(\frac{pos}{10000^{\left(\frac{2i}{d_{model}}\right)}}\right) \tag{4.6}$$

where $pos$ is the position and $i$ is the dimension of the sequence.

### 4.2.3 Transformer Encoder-Decoder Architecture for WGAN-gp *generator*

Figure 4.4 illuminates the encoder decoder architecture of the Transformer, adapted from Vaswani et al. [24]. Encoder and Decoder both consists of $N$ identical layers, respectively. The number of layers/depth for encoder or decoder can be adjusted as per the task at hand. The role of the encoder is to map attention-based representation of input vectors to latent vectors. This latent vector is input to decoder which serves as a memory from the input sequence. Decoder have similar architecture like Encoder with additional masking mechanism which prevents Decoder from learning future information.

#### Encoder

The encoder consists of $N$ identical layers. Each layer has two sub layers - a multi-head attention layer and a point-wise fully connected layer. The point-wise fully connected layer applies two linear transformations with a ReLU activation in between.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{4.7}$$

Each sub-layer uses a residual connection [59] with layer normalization. It is computed as $LayerNorm(x+SubLayer(x))$, where $x$ is input to the layer.

### Decoder

The decoder also consists of $N$ identical layers with similar architecture to the encoder with an additional Masked Multi-Head Attention layer. Masked Multi-Head Attention layer is added to prevent the decoder from attending to future information. This implies that for a prediction task, predicted output at position $t_i$ can depend on known outputs $t_{1,2,...,t_{i-1}}$. The multi-head attention layer in decoder computes attention on output from encoder stack as *keys* and *values* and its preceding sub-layer Masked Multi-Head Attention layer as *query*. Similar to the encoder, each sub-layer uses a residual connection [59] with layer normalization.

### Architecture of Wasserstein Adversarial Transformer

WGAN-gp Transformer *generator* (illustrated in Figure 4.4) is design to have one layer of encoder and subsequent one layer of decoder. For cloud workload forecasting, experiments show that architecture design with one encoder layer and one decoder layer for *generator* yields optimal results. To embed the positional information in input, positional encoding explained in section 4.2.2 is performed on input $x_{0:t_k}$, where $x$ is value from time step $0...k$, $k \in \mathbb{R}$ . Similarly, positional encoding is performed on input to decoder. Input to decoder is $x_{t_k}$, which is last time step value of the input sequence to encoder. The objective of this model is to predict job arrival rate value at next time step $k+1$ from input sequence of $k$ time steps.

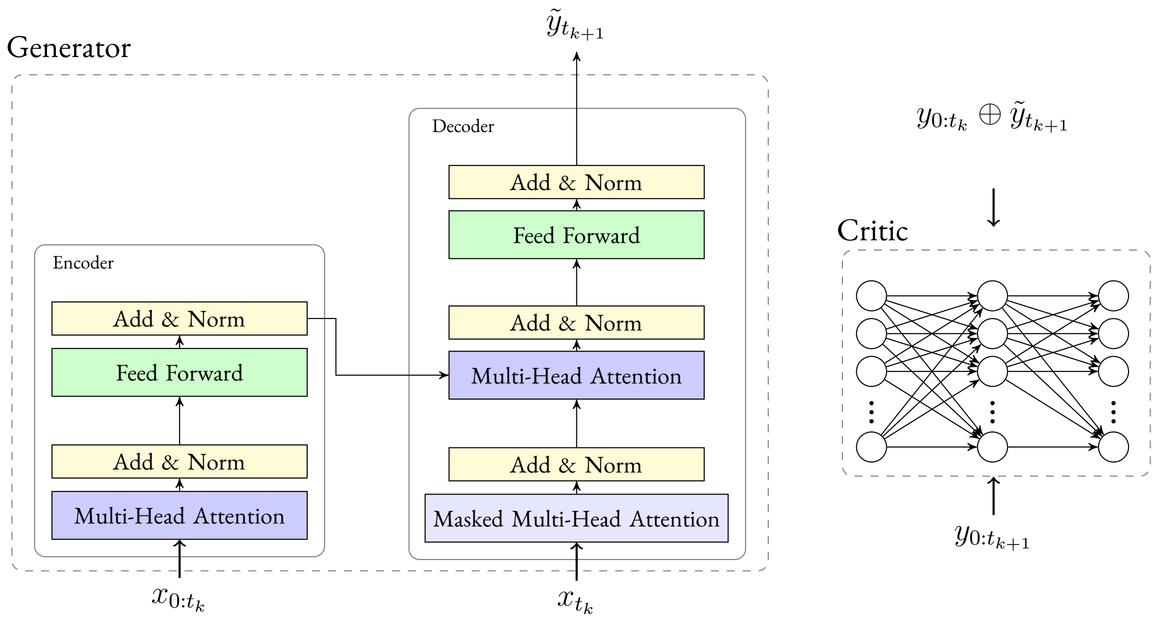


Figure 4.4: *generator* and *critic* in WGAN-gp Transformer model

## 4.3 Wasserstein Adversarial Transformer model

Building on the foundation of WGANs and Transformer Network, this work proposes Wasserstein Adversarial Transformer (WGAN-gp Transformer) network to tackle the problem of cloud workload forecasting. Time series forecasting models are designed to optimize specific objective alone, such as mean absolute error or quantile loss function. Initially, Transformer network is trained to optimize the mean absolute error loss function. However, the Transformer network alone fails to map the dynamic nature of the cloud workload forecast to provide accurate predictions. This thesis proposes a Wasserstein Adversarial Transformer network with gradient penalty for cloud workload prediction to improve prediction accuracy. In order to do so, we have used a *critic* $D$, attached on top of the Transformer *generator*. In the original work Gulrajani et al. [51] proposed Algorithm 1.

The *generator* is based on the encoder-decoder architecture of the vanilla version of the Transformer network and three fully connected linear layers with as *critic*. The encoder encodes the input sequence vector $[\mathbf{X_{1:k}}]$ to latent vector $[\tilde{\mathbf{X}}_{\mathbf{1:k}}]$. This latent vector $[\tilde{\mathbf{X}}_{\mathbf{1:k}}]$ serves as memory to the decoder to generate prediction for next time step $k+1$, from the encoded input sequence. Adversarially, the Transformer network, as a *generator* $\mathbf{G}$, attempts to minimize the mean absolute error ($\mathcal{L}_{\text{MAE}}$) between prediction sequence and ground truth and Wasserstein loss for the generator in WGAN-gp algorithm [51]. In order to train the *generator*, below is the updated loss function:

$$\underset{G}{min}\,\mathbb{E}_{y\sim\mathbb{P}_r(y|x)}[(|G(x)-y|)] - \mathbb{E}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] \tag{4.8}$$

where $\mathbb{P}_g$ is *generator* distribution, $\mathbb{P}_r$ is real data distribution, $y$ is ground truth output for input $x$ to *generator*.

The *critic* is a network of three fully connected linear layers with LeakyReLU [60] as activation function [50][61]. LeakyReLU controls the angle of negative slope to allow neuron to fire again in case of small gradient flow. Thus for *critic* architecture, LeakyReLU is computed by $f(x) = max(\alpha x, x)$, for $\alpha = 0.2$. In the WGAN-gp algorithm, the *critic*'s objective function is same as equation 4.1.

**Algorithm 2:** Wasserstein GAN with gradient penalty for cloud workload forecasting.

```
Require: $\lambda$, the gradient penalty coefficient. m, the batch size.
         $n_{critic}$, the number of iterations of the critic per
         generator iteration. $m_{MADGRAD}$, momentum value.
Require: initial critic parameters $w_0$, initial generator
         parameters $\theta_0$.
while $\theta$ has not converged do
    for $t = 1, ...., n_{critic}$ do
        for $i = 1, ...., m$ do
            Sample real data $x \sim \mathbb{P}_r$, a latent variable $z \sim p(z)$, a
             random number $\epsilon \sim U[0,1]$.
            $\tilde{x} \leftarrow G_\theta(z)$
            $\hat{x} \leftarrow \epsilon x + (1-\epsilon)\tilde{x}$
            $L^{(i)} \leftarrow D_w(\tilde{x}) - D_w(x) + \lambda(||\nabla_{\hat{x}} D_w(\hat{x})||_2 - 1)^2$
        end
        $w \leftarrow$ MADGRAD ($\nabla_w \frac{1}{m}\sum_{i=1}^{m} L^{(i)}, w, \alpha, m_{MADGRAD}$)
    end
    Sample a batch of latent variables $\{z^{(i)}\}_{i=1}^{m} \sim p(z)$
    $\theta \leftarrow$ MADGRAD($\nabla_\theta \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}_{\text{MAE}} - D_w(G_\theta(z)), \theta, \alpha, m_{MADGRAD}$)
end
```

### Model Optimizer

Adam optimization method [62] is widely used and heavily adopted for deep learning optimization, especially for training GANs. The task of optimizer is to perform optimization by gradient step for network parameters. In the original proposed WGAN-gp algorithm [51], Adam optimizer is used to train both *generator* and *critic*. However, the Adam optimization method often converges to bad local minima for some major tasks discussed in Wilson et al. [63], highlighting the weak ability of adaptive methods generalizing poorly. With a goal of designing a general-purpose deep learning optimizer, Defazio et al. [27] proposed a **M**omentumized, **A**daptive, **D**ual averaged **GRAD**ient (MADGRAD) Method for Stochastic Optimization [26]. MADGRAD is optimization method which computes adaptive learning rate for each parameter. MADGRAD is based upon the dual averaging formulation of AdaGrad. This work use MADGRAD to optimize *generator* and *critic* in the proposed Wasserstein Adversarial Transformer model (WGAN-gp Transformer).

### Adversarial Training

The *critic* is updated first and then *generator* is updated learning from *critic*. Algorithm 2 illustrates training algorithm of the proposed model. The *generator* is trained with updated loss function introduced in equation 4.8. Instead of Adam optimizer, we have proposed to use MADGRAD optimizer which showcase effective results for training WGAN-gp Transformer model. For training of Wasserstein GAN with gradient penalty (WGAN-gp), default values are set for $n_{critic} = 5, \lambda = 10$, learning rate $\alpha = 0.001$, and MADGRAD optimizer parameters $m = 0.9$ and default weight decay as zero. Refer algorithm 2 for WGAN-gp Transformer for cloud workload forecasting.

Grid search is performed for the searhing the optimal hyperparameters for training WGAN-gp Transformer. The model size $d_{model}$ unit used for training *generator* is same number of input features in *critic* Linear layers. For the *generator*, below are the hyperparameters:

- $d_{model}$ - the model size; positional encoding size.
- $n_{head}$ - the number of heads in multi-head attention layer in encoder and decoder.

- $m$ - batch size
- $n$ - the length of the input sequence

The proposed model will be evaluated on various cloud workloads data sets to predict job arrival rate at the next time step. WGAN-gp Transformer shows optimal results for next time step prediction we will discuss the evaluation results in the next Chapter.

# Chapter 5

# Experiments

This Chapter details about the experimental evaluation of WGAN-gp Transformer. The evaluations focus on measuring the accuracy and computational overhead of the WGAN-gp Transformer compared to baseline LoadDynamics, which is based on LSTM [19]. This work shows a comprehensive experimental evaluation of WGAN-gp Transformer across many real-world cloud workload data sets from Google, Facebook, Wiki, Azure, and Alibaba.

## 5.1 Experimental Setup

Real-world cloud workloads are represented as time series data exhibit correlated workload patterns [40]. Datasets evaluated in this work are univariate time series representing job arrival rates, which is defined as the number of jobs arrived at a specified time unit to cloud systems. In this section, we briefly discuss individual datasets and evaluation metrics for the conducted experiments. For comparison, we have used LSTM-based DynaLoad [19] as our baseline.

### Cloud Workload Datasets

Cloud workloads collected from different categories are used to evaluated WGAN-gp Transformer. Wikipedia datatset [64] is web application trace. Wikipedia trace shows a strong seasonal pattern sampled at time intervals of 10 mins and 30 mins. Facebook workload [65] is data center workload that shows dynamic spikes and high fluctuations in the job arrival rate. The Facebook dataset is a smaller dataset with a workload trace accounting for a 24-hour period. Thus, Facebook workload trace is evaluated at 5 and 10 mins intervals (shown in Figure 5.1). In addition to workload traces evaluated in Jayakumar et al. [19], we have added three new datasets in the evaluation - public cloud workload from Azure released in 2019[1], Azure Functions trace[2] [66], and Alibaba 2018 cluster trace[3]. Alibaba 2018 dataset contains eight day long cluster trace, and evaluated time intervals are 5 mins and 10 mins. Azure public cloud dataset and Azure functions trace, released in 2019, both are 30 days long traces and evaluated at 30 mins and 60 mins time interval.

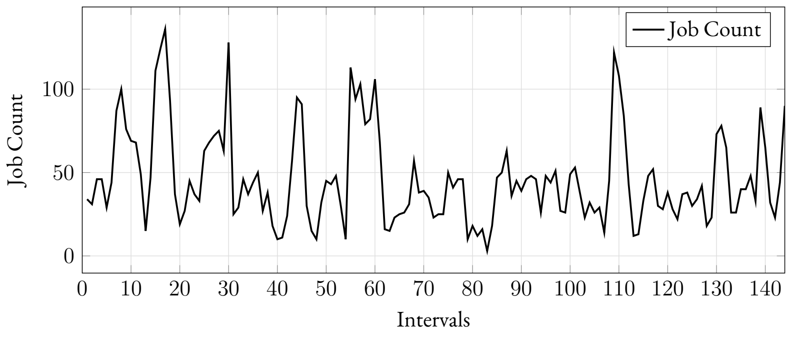


Figure 5.1: Facebook Trace (10 minutes interval)

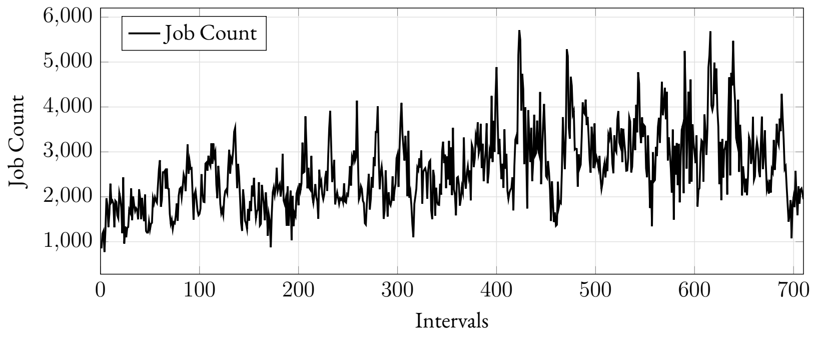


Figure 5.2: Azure Public Cloud 2018 Trace (60 minutes interval)

Different time granularity may exhibit subtle variations in the time series patterns. Thus, each dataset is evaluated at various time intervals to gauge the proposed WGAN-gp Transformer's performance. The time intervals at which individual dataset is studied is illustrated in Table 5.1. The effect of various time

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

[2]https://github.com/Azure/AzurePublicDataset/blob/master/AzureFunctionsDataset2019.md

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

intervals for respective workload is discussed in next section. For the rest of this thesis, we follow similar annotation like Jayakumar et al. [19], to define *workload configuration* as a workload with a specific time interval.

Table 5.1: Cloud Workload Datasets for evaluated time intervals

| **Workload** | **Time intervals (in $mins$)** | **Type of Dataset** |
|---|---|---|
| Facebook (FB) | $5, 10$ | Data Center |
| Alibaba 2018 | | Data Center |
| Google (GL) | $10, 30$ | Data Center |
| Wiki (wiki) | | Web |
| Azure 2017 VM workload | $10, 30, 60$ | Public Cloud |
| Azure 2019 VM workload | $30, 60$ | Public Cloud |
| Azure Functions 2019 | | Public Cloud |

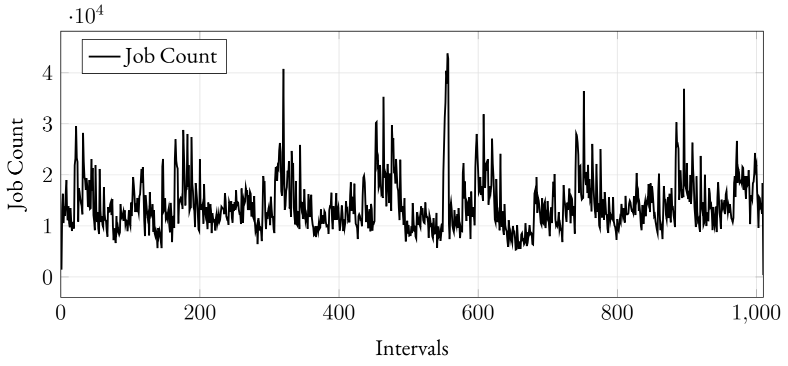


Figure 5.3: Alibaba 2018 Trace (10 minutes interval)

## Evaluation metric

We have used Mean Absolute Percentage Error as evaluation metric to assess the proposed method against baseline. Mean Absolute Percentage Error is calculated as,

$$\texttt{MAPE} = 100 * \left(\frac{1}{n}\right) \sum_{i=1}^{n} \left| \frac{\tilde{y}_i - y_i}{y_i} \right| \tag{5.1}$$

where $n$ is the total number of data points, $\tilde{y}_i$ represent predicted value at time step $i$ and $y_i$ represents actual value at time step $i$.

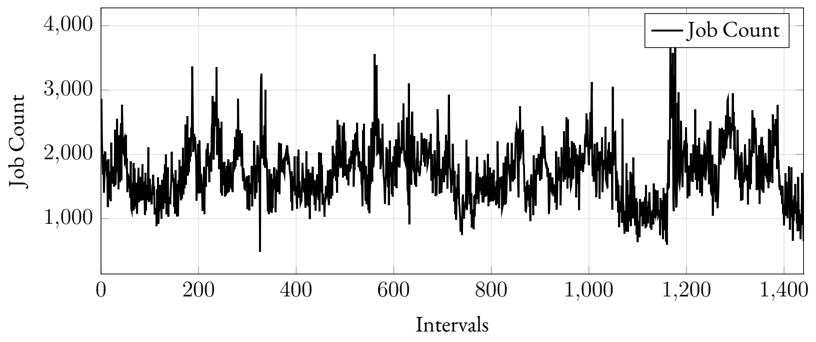


Figure 5.4: Azure Public Cloud 2019 Trace (30 minutes interval)

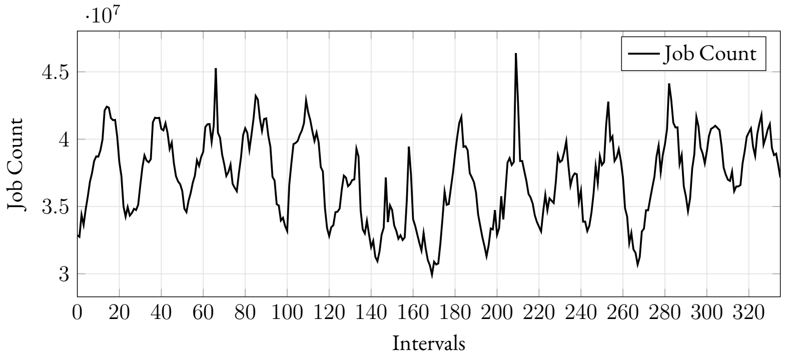


Figure 5.5: Azure Functions 2019 Trace (60 minutes interval)

**Baseline**

To measure the performance of WGAN-gp Transformer, we compare proposed model on the discussed seven datasets (shown in Table 5.1) with baseline model, i.e, LoadDynamics (LSTM) [19](shown in Table 5.3), which is state-of-the-art recurrence based time series forecasting model. LoadDynamics employs LSTM model to automatically optimize LSTM for individual workload using Bayesian Optimization. As our baseline, we use the brute force approach of the LoadDynamics which performs hyperparamter search for LSTM in predefined hyperparameter search space. For LoadDynamics, the accuracy of the model is determined by hyperparameter values of LSTM model such as number of LSTM layers, the memory cell $C$ size, and the history length (input sequence length $n$). For baseline, we have established same setup described for evaluating LoadDynamics [19], in addition to new cloud workloads to be evaluated. The

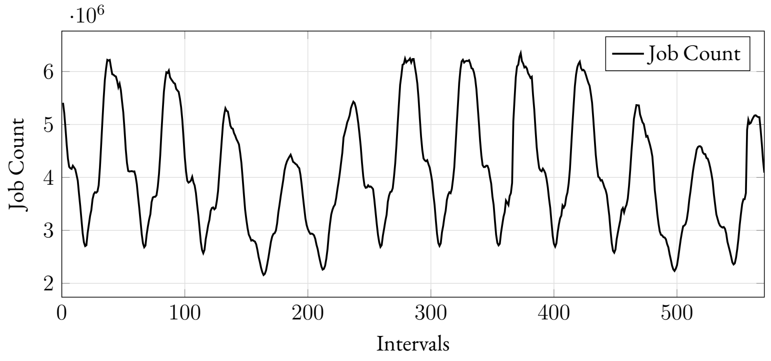


Figure 5.6: Wikipedia Trace (30 minutes interval)

hyperparameter search space for baseline LSTM model is illustrated in Table 5.2. The training and inference is performed on same machine as we evaluate WGAN-gp Transformer and are fine-tuned on a single GPU.

Table 5.2: Hyperparameter search space baseline model LSTM [19]

| **Workload** | **History Length $n$** | **batch size $m$** | **LSTM Layers** | **C size** |
|---|---|---|---|---|
| Facebook (FB) | $[3-46]$ | $[16-256]$ | $[1-5]$ | $[1-50]$ |
| Azure Functions 2019 | $[7-108]$ | $[16-512]$ | | $[1-100]$ |
| Google (GL) | $[28-676]$ | $[16-1024]$ | | |
| Azure 2017 VM workload | $[14-682]$ | | | |
| Wiki (wiki) | $[12-274]$ | | | |
| Alibaba 2018 | $[20-324]$ | | | |
| Azure 2019 VM workload | $[14-230]$ | | | |

## Implementation

WGAN-gp Transformer is implemented in PyTorch[4] and Scikit-learn[5]. The baseline model and WGAN-gp Transformer are trained and evaluated on a machine with a single GPU (NVIDIA GeForce RTX 2080 Ti [6]). The configuration of WGAN-gp Transformer is shown in Table 5.4. The Transformer network alone is trained on "mean absolute error" as the loss function. In WGAN-gp Transformer, we improve

[4]https://pytorch.org/docs/stable/index.html
[5]https://scikit-learn.org/
[6]https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2080-ti/

Table 5.3: Evaluated time series forecasting models

| Prediction Model | Description |
|---|---|
| LSTM (Baseline) | The LSTM [20] model leverages recurrences to map an input sequence to an output observation at next time step. |
| Transformer Encoder-Decoder | Transformer network is encoder-decoder architecture which employs attention mechanism. The encoder encodes the input sequence to generate $d_{model}$ dimension latent vector. This latent vector serves as memory to decoder to generate an output observation at next time step. |
| WGAN-gp Transformer (Proposed model) | The proposed model WGAN-gp Transformer is based on Wasserstein GAN and Transformer network. WGAN-gp Transformer improves upon the prediction done by Transformer Encoder-Decoder model, since Transformer Encoder-Decoder alone is unable to learn the complexities of cloud workloads. |

upon Transformer network ability by applying improved WGAN algorithm with gradient penalty [51]. To train *generator* and *critic* in WGAN-gp Transformer, we use MADGRAD Optimization method [27]. The MADGRAD optimizer is used with default values ($m = 0.9$ momentum value, $weight\_decay = 0$, $eps = 1 - e6$). The learning rate is $\alpha = 0.001$. WGAN-gp Transformer is trained for $1000$ epochs, which is proved to be sufficient across all cloud workload datasets to achieve convergence.

For an entire JAR sequence of cloud workload dataset, the sequence is divided into a training set of 60%, 20% for cross-validation, and the last 20% is tested to evaluate the trained model accuracy, similar to baseline [19]. To prepare the data for input to WGAN-gp Transformer, we apply the sliding window approach to divide the data into sequences of length $n$. The parameter history length $n$ for comparison is based on baseline method [19]. 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. Hyperparameters for

WGAN-gp Transformer are history length $n$, batch size $m$, the model size $d_{model}$ for both *generator* and *critic*, and the number attention heads $n_{head}$ (illustrated in Table 5.4).

Table 5.4: Hyperparameter search space for *generator* in WGAN-gp Transformer

| **Workload** | **History Length** $n$ | **batch size** $m$ | $d_{model}$ | $n_{head}$ |
|---|---|---|---|---|
| Facebook (FB) | $[3-46]$ | $[16-256]$ | $[8, 16, 32, 64, 128, 512]$ | $[4, 8]$ |
| Azure Functions 2019 | $[7-108]$ | $[16-512]$ | | |
| Google (GL) | $[28-676]$ | $[16-1024]$ | | |
| Azure 2017 VM workload | $[14-682]$ | | | |
| Wiki (wiki) | $[12-274]$ | | | |
| Alibaba 2018 | $[20-324]$ | | | |
| Azure 2019 VM workload | $[14-230]$ | | | |

Table 5.5: Results for evaluated models for cloud workloads; Metric : MAPE; **WGAN-gp**: Wasserstein Generative Adversarial Network with gradient penalty [51]

| **Workload** | **LSTM** | **Transformer** | **WGAN-gp Transformer with ADAM** | **WGAN-gp Transformer with MADGRAD** |
|---|---|---|---|---|
| GL-10m | 11.49 | 17.13 | 14.03 | **10.58** |
| GL-30m | 9.12 | 11.50 | 10.24 | **8.34** |
| Azure-2017-VM-10m | 42.63 | 43.63 | 41.58 | **41.32** |
| Azure-2017-VM-30m | 29.35 | 40.64 | 36.59 | **27.48** |
| Azure-2017-VM-60 | 16.11 | 20.26 | 16.54 | **12.77** |
| FB-5m | 47.20 | 64.85 | 59.49 | **42.11** |
| FB-10m | 43.68 | 82.35 | 48.51 | **39.31** |
| Wiki-10m | **1.17** | 1.38 | 1.21 | 1.34 |
| Wiki-30m | **1.75** | 4.54 | 3.11 | 3.43 |
| Azure-2019-VM-30m | 19.74 | 19.40 | 16.43 | **15.19** |
| Azure-2019-VM-60m | 13.5 | 14.46 | 11.17 | **10.82** |
| Azure-Func-2019-30m | **1.63** | 7.42 | 6.74 | 3.05 |
| Azure-Func-2019-60m | 2.06 | 2.32 | 2.31 | **1.85** |
| Alibaba-2018-5m | 17.95 | 17.42 | 17.32 | **15.76** |
| Alibaba-2018-10m | 16.90 | 25.55 | 15.08 | **14.67** |

## 5.2 WGAN-gp Transformer Evaluation

Table 5.5 illustrates the prediction error for the proposed WGAN-gp Transformer and baseline method LSTM on evaluated workload configurations (shown in Table 5.1). These forecast prediction errors are calculated for the test set. Reported prediction errors imply the variation in forecasting performance when datasets are sampled at various time intervals. For larger time intervals, majority of evaluated workloads provide comparatively accurate results for WGAN-gp Transformer and LSTM. However, for datasets indicating seasonal pattern like Wikipedia and Azure Functions Datasets (shown in Figure 5.5, 5.6), smaller sample rate show low prediction error providing higher accuracy. Overall, WGAN-gp Transformer provide up to 5.1% lower MAPE, compared to LSTM.

WGAN-gp Transformer functions on attention mechanism to extract information from input sequence for prediction task unlike LSTM, which uses recurrences. For training each workload configuration, a small sample rate provides more time intervals, increasing the dataset size to be evaluated. Larger dataset tend to need more training time with WGAN-gp Transformer compared to LSTM. For instance, Facebook is fairly small dataset accounting for 24-hour period trace. Facebook 10 minutes workload configuration takes 4.75 hours to find the optimal set of hyperparameters from the defined set (illustrated in Table 5.4). On the contrary, LSTM takes 1.78 hours to train entire workload configuration (shown in Table 5.2). Especially for Facebook-10m, once found optimal set of hyperparameters WGAN-gp Transformer takes 2ms for inference, which 10 times faster than LSTM making an inference in 23ms. Table 5.6 illuminates the time inference difference between LSTM and WGAN-gp Transformer[7]. The inference time is calculated for the entire test dataset for respective workload configuration. On average, LSTM inference time is 25.57ms and WGAN-gp Transformer makes an inference in 4.85ms. The evaluation results show that WGAN-gp Transformer is 5 times faster than state-of-the-art LSTM to make an inference.

Figure 5.8 illuminates that WGAN-gp Transformer performs consistently better to provide more accurate predictions for evaluated cloud workloads, except for Wikipedia and Azure Functions 2019 Dataset (30 mins interval). Even though WGAN-gp Transformer achieves lower prediction error for Wikipedia

[7] average inference time is calculated for all workload configurations for respective datasets evaluated in this work.

Table 5.6: Inference time comparison for cloud workloads

| Workload | LSTM | WGAN-gp Transformer |
|---|---|---|
| Google | 27ms | 11ms |
| Facebook | 23ms | 2ms |
| Wiki | 21ms | 5ms |
| Azure 2018 | 26ms | 5ms |
| Azure 2019 | 28ms | 4ms |
| Azure Functions 2019 | 19ms | 4ms |
| Alibaba 2018 | 29ms | 3ms |

and Azure Functions 2019 Dataset, LSTM report lower prediction error 0.17%, 1.68%, and 0.9% for Wiki-10m, Wiki-30m, and Azure-Func-2019-60m workload configurations respectively.

### Detailed evaluation of WGAN-gp Transformer model

To study the implication of individual element of WGAN-gp Transformer we evaluated vanilla version of Transformer network [24] and WGAN-gp Transformer with Adam optimizer [51] for 15 workload configurations. The evaluation results are illustrated in Table 5.9. WGAN-gp Transformer has consistently performed better than Transformer and WGAN-gp Transformer when optimized with Adam optimizer [62]. For some workload configurations WGAN-gp Transformer optimized with Adam and WGAN-gp Transformer performance error difference up to 9.2%. However, in case of Wiki-30m, WGAN-gp Transformer with Adam performed comparatively better than WGAN-gp Transformer by 0.32%, but still performed poor compared to LSTM by 1.36%.

### WGAN-gp Transformer Evaluation with Auto-scaling

WGAN-gp Transformer is applied to an auto-scaling policy that managed the VMs executing in the Google cloud, to gauge the performance and resource efficiency. The evaluations are done on subset of workload to compare the performance of state-of-the-art LSTM and WGAN-gp Transformer. Figure 5.7 illustrates the process to evaluate auto-scaling. The workload predictor result for the predicted time interval is provided to job scheduler. Assume for next time interval $i^{th}$, $P_i$ jobs are predicted at $(i-1)^{th}$

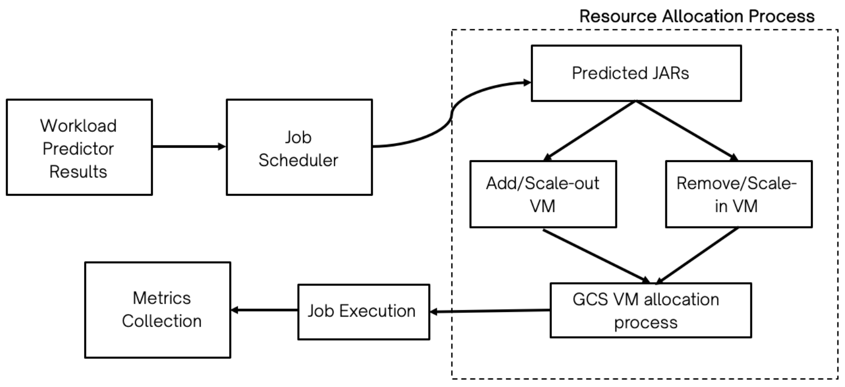


Figure 5.7: Auto-scaling mechanism

interval representing number of Virtual Machines (VM)s to be created in advance. One VM per predicted job $P_i$ is allocated. Assume $T_i$ is the actual number of jobs arriving at time interval. If $(T_i > P_i)$, then it results in under-provisioning and to accommodate extra demand of the jobs more VMs will be allocated. In this case, additional time will be needed to finish the jobs due the VM startup time. On contrary, if $(T_i < P_i)$, this results in over-provisioning and incur extra unnecessary cost with the VMs running idle.

Google Cloud's *e2-medium* VM instances are used for this auto-scaling evaluation. Facebook and Azure 2019 workloads are evaluated for compare auto-scaling performance. To evaluate Facebook workload configuration, Cloud Suite's *Data Analytics* benchmark is used which consists of running a Naive Bayes classifier on a Wikimedia dataset. This benchmark addresses analyzing large amounts of machine learning tasks in datacenters using MapReduce framework [67–69]. For Facebook dataset (shown in Table 5.8), evaluation results show significant reduction in under-provisioning by $27.95\%$. However, WGAN-gp Transformer over-provisions with over $5.8\%$ more than LSTM. Cloud Suite's *In-Memory Analytics* benchmark is used to evaluate Azure 2019 workload configuration as the jobs to execute, simulating a system service machine-learning training and inference requests. *In-Memory Analytics* benchmark uses Apache Spark and execute collaborative filtering algorithm in-memory on dataset of user-movie ratings [68–71]. JARs are scaled down by 100 times to reduce the number of VMs created to be less than 50, only for Azure workload. The scale down of JARs does not affect the prediction accuracy for the evaluated prediction techniques. For Azure 2019 dataset, Table 5.7 illustrates that the under-provisioning and over-provisioning is reduced by 2.56% and 1.92%.

Table 5.7: Performance evaluation, VM under-provisioning rates and over-provisioning rates for Azure 2019 dataset

| System | under-provisioning rate(%) | over-provisioning rate(%) |
|---|---|---|
| LSTM | 9.63 | 8.60 |
| WGAN-gp Transformer | 7.07 | 6.68 |

Table 5.8: Performance evaluation, VM under-provisioning rates and over-provisioning rates for Facebook dataset

| System | under-provisioning rate(%) | over-provisioning rate(%) |
|---|---|---|
| LSTM | 40.22 | 10.33 |
| WGAN-gp Transformer | 12.27 | 16.13 |

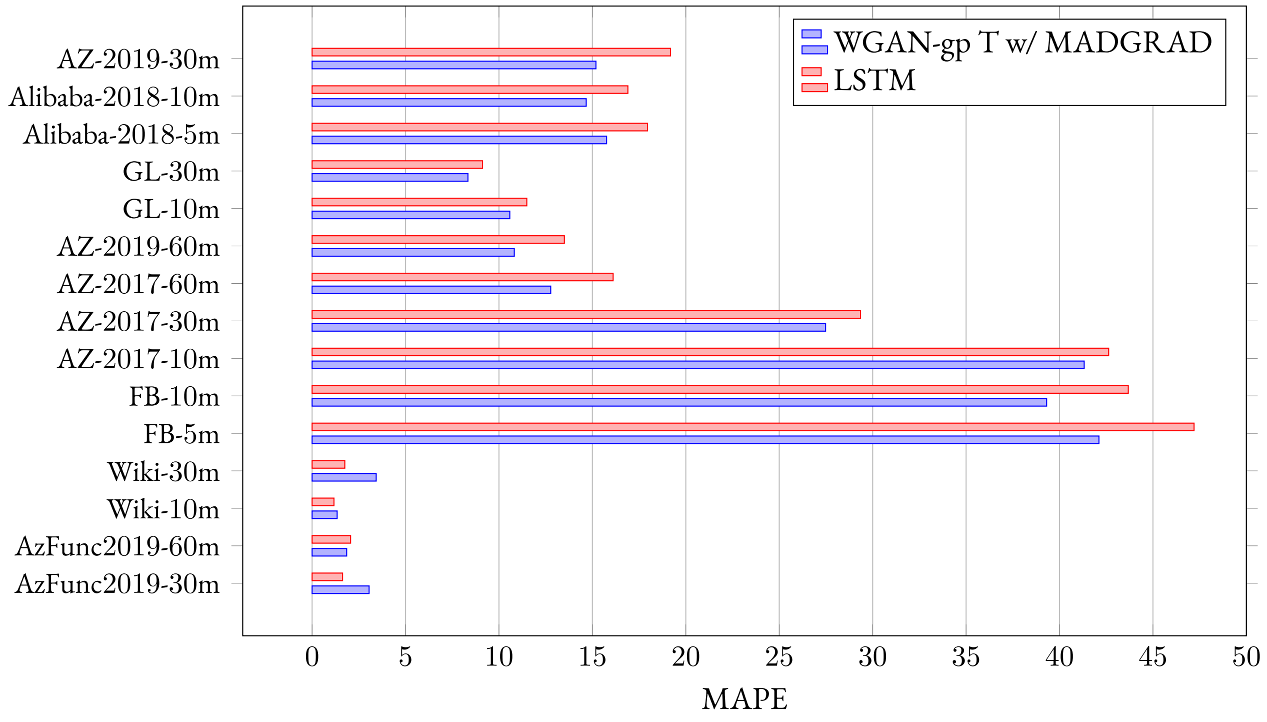


Figure 5.8: Prediction test error comparison with baseline

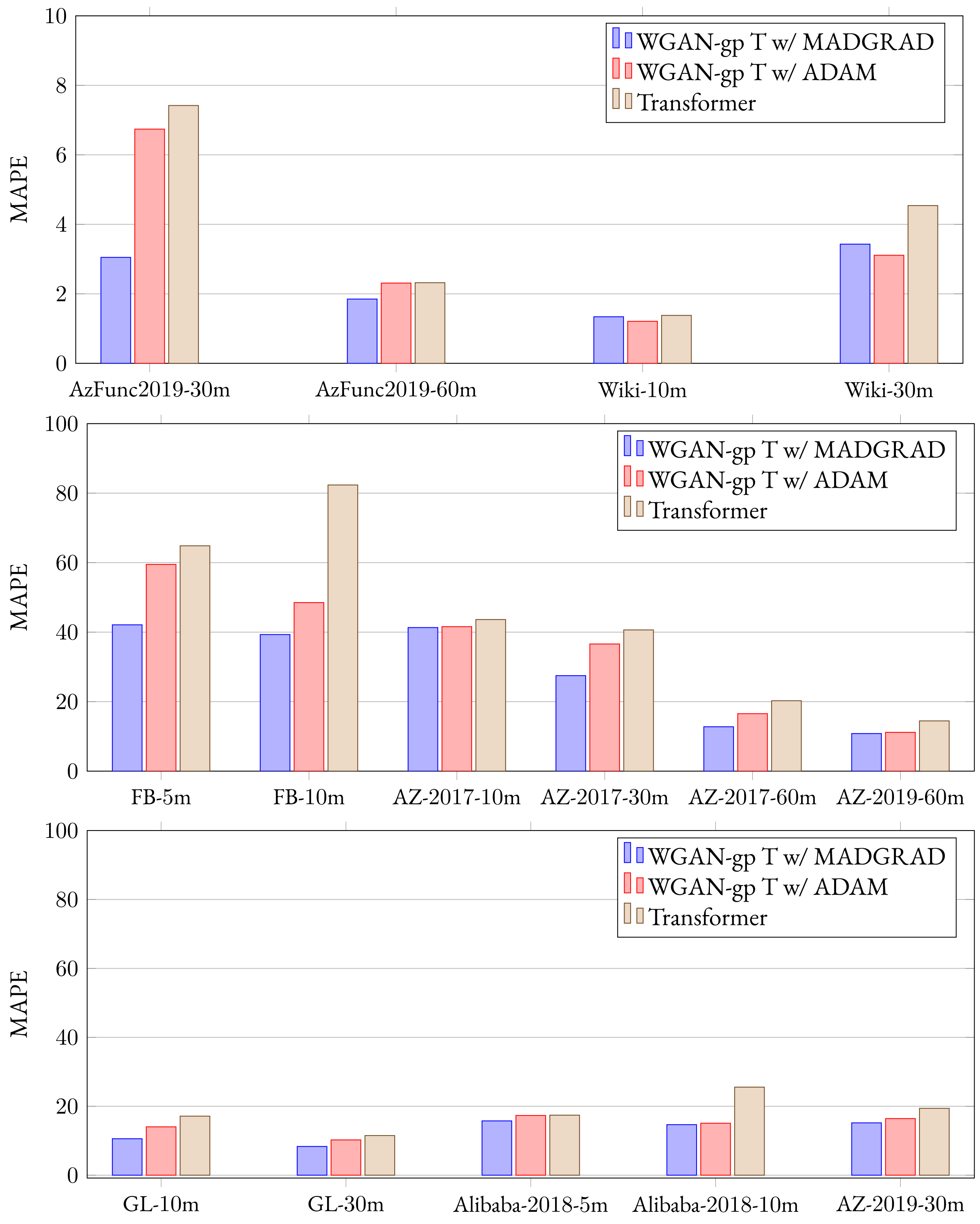


Figure 5.9: Prediction test error comparison for detailed analyis

# Chapter 6

# Conclusion

Accurately forecasting user request rates (e.g., job arrival rates) benefits optimizing cloud operating costs and guaranteeing application performance goals through proactive cloud autos-caling. To addresses the problem of job arrival prediction for dynamic cloud workloads, this thesis proposed a novel time series forecasting method, called WGAN-gp Transformer, based on Transformer network and Wasserstein Generative Adversarial networks. When trained with an improved Wasserstein Generative Network algorithm, the Transformer network optimally models dynamic patterns in cloud workload data. The proposed method was evaluated on 15 diverse real-world workload configurations and achieved a lower prediction error rate up to $5.1\%$. Evaluation results indicate that attention-based models are better at capturing relevant information from sequences of varying lengths to make a prediction. Furthermore, WGAN-gp Transformer significantly reduced inference time by 5 times compared to the state-of-the-art LSTM-based model. Auto-scaling evaluations also showed that the over-provisioning and under-provisioning rates with WGAN-gp Transformer were reduced by 2.56% and 1.92% for Facebook and Azure 2019 workloads, respectively. By mitigating the limitations of existing predictors for cloud workloads, this work would benefit cloud applications to satisfy their performance goals with more accurate predictive auto-scaling.

# Bibliography

[1] Zhiming Shen et al. "Cloudscale: elastic resource scaling for multi-tenant cloud systems". In: *Proceedings of the 2nd ACM Symposium on Cloud Computing*. 2011, pp. 1–14.

[2] Timothy Wood et al. "Profiling and modeling resource usage of virtualized applications". In: *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer. 2008, pp. 366–387.

[3] Ming Mao and Marty Humphrey. "Auto-scaling to minimize cost and meet application deadlines in cloud workflows". In: *International Conference on High Performance Computing Networking, Storage and Analysis (SC' 11)*. Seattle, WA, USA, Nov. 2011.

[4] Krzysztof Rzadca et al. "Autopilot: workload autoscaling at Google". In: *Fifteenth EuroSys Conference 2020 (EuroSys '20)*. Heraklion, Greece, Apr. 2020.

[5] Ming Mao and Marty Humphrey. "A Performance Study on the VM Startup Time in the Cloud". In: *2012 IEEE Fifth International Conference on Cloud Computing (IEEE CLOUD 2012)*. Honolulu, HI, USA, June 2012.

[6] In Kee Kim et al. "Empirical Evaluation of Workload Forecasting Techniques for Predictive Cloud Resource Scaling". In: *9th IEEE International Conference on Cloud Computing (IEEE CLOUD' 16)*. San Francisco, CA, USA, June 2016.

[7] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. "PRESS: PRedictive Elastic ReSource Scaling for cloud systems". In: *6th International Conference on Network and Service Management (CNSM '10)*. Niagara Falls, Canada, Oct. 2010.

[8] Nilabja Roy, Abhishek Dubey, and Aniruddha Gokhale. "Efficient autoscaling in the cloud using predictive models for workload forecasting". In: *2011 IEEE 4th International Conference on Cloud Computing*. IEEE. 2011, pp. 500–507.

[9] Neeraja J. Yadwadkar, Ganesh Ananthanarayanan, and Randy Katz. "Wrangler: Predictable and Faster Jobs using Fewer Resources". In: *ACM Symposium on Cloud Computing (SoCC)*. 2014.

[10] Joao Loff and Joao Garcia. "Vadara: Predictive Elasticity for Cloud Applications". In: *IEEE International Conference on Cloud Computing Technology and Science (CloudCom '14)*. Singapore, Dec. 2014.

[11] Yexi Jiang et al. "ASAP: A Self-Adaptive Prediction System for Instant Cloud Resource Demand Provisioning". In: *IEEE International Conference on Data Mining (ICDM '11)*. Vancouver, BC, Canada, Dec. 2011.

[12] Rodrigo N. Calheiros et al. "Workload Prediction Using ARIMA Model and Its Impact on Cloud Applications' QoS". In: *IEEE Trans. on Cloud Computing* 3(4) (2015).

[13] Eli Cortez et al. "Resource Central: Understanding and PredictingWorkloads for Improved Resource Management inLarge Cloud Platforms". In: *ACM Symp. on Operating Systems Principles*. 2017.

[14] In Kee Kim et al. "Cloudinsight: Utilizing a council of experts to predict future cloud application workloads". In: *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*. IEEE. 2018, pp. 41–48.

[15] Siddhant Kumar et al. "Association Learning based Hybrid Model for Cloud Workload Prediction". In: *Int'l Joint Conf. on Neural Networks*. 2018.

[16] Chanh Nguyen, Cristian Klein, and Erik Elmroth. "Multivariate LSTM-Based Location-Aware Workload Prediction for Edge Data Centers". In: *Int'l Symp. on Cluster, Cloud and Grid Computing*. 2019.

[17] Qingchen Zhang et al. "An Efficient Deep Learning Model to PredictCloud Workload for Industry Informatics". In: *IEEE Trans. on Industrial Informatics* 14 (7 2018).

[18] In Kee Kim et al. "Forecasting cloud application workloads with CloudInsight for predictive resource management". In: *IEEE Transactions on Cloud Computing* ().

[19] Vinodh Kumaran Jayakumar et al. "A Self-Optimized Generic Workload Prediction Framework for Cloud Computing". In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE. 2020, pp. 779–788.

[20] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9(8) (1997).

[21] Sadeka Islam, Srikumar Venugopal, and Anna Liu. "Evaluating the impact of fine-scale burstiness on cloud elasticity". In: *ACM Symposium on Cloud Computing (SoCC '15)*. Kohala Coast, Hawaii, USA, Aug. 2015.

[22] Charles Reiss et al. "Towards understanding heterogeneous clouds at scale: Google trace analysis". In: *Intel Science and Technology Center for Cloud Computing, Tech. Rep* 84 (2012).

[23] John Wilkes. *More Google cluster data*. Google research blog. Posted at `http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html`. Mountain View, CA, USA, Nov. 2011.

[24] Ashish Vaswani et al. "Attention is all you need". In: *arXiv preprint arXiv:1706.03762* (2017).

[25] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks". In: *International conference on machine learning*. PMLR. 2017, pp. 214–223.

[26] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011).

[27] Aaron Defazio and Samy Jelassi. "Adaptivity without Compromise: A Momentumized, Adaptive, Dual Averaged Gradient Method for Stochastic Optimization". In: *arXiv preprint arXiv:2101.11075* (2021).

[28] Eyal Zohar, Israel Cidon, and Osnat Mokryn. "The power of prediction: Cloud bandwidth and cost reduction". In: *ACM SIGCOMM Computer Communication Review* 41.4 (2011), pp. 86–97.

[29] Norman Bobroff, Andrzej Kochut, and Kirk Beaty. “Dynamic placement of virtual machines for managing SLA violations”. In: *2007 10th IFIP/IEEE International Symposium on Integrated Network Management*. IEEE. 2007, pp. 119–128.

[30] Haibo Mi et al. “Online self-reconfiguration with performance guarantee for energy-efficient large-scale cloud computing data centers”. In: *2010 IEEE International Conference on Services Computing*. IEEE. 2010, pp. 514–521.

[31] Andrew Krioukov et al. “Napsac: Design and implementation of a power-proportional web cluster”. In: *Proceedings of the first ACM SIGCOMM workshop on Green networking*. 2010, pp. 15–22.

[32] Peter A Dinda and David R O’Hallaron. “Host load prediction using linear models”. In: *Cluster Computing* 3.4 (2000), pp. 265–280.

[33] Gong Chen et al. “Energy-Aware Server Provisioning and Load Dispatching for Connection-Intensive Internet Services.” In: *NSDI*. Vol. 8. 2008, pp. 337–350.

[34] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. “Dynamic resource allocation for shared data centers using online measurements”. In: *International Workshop on Quality of Service*. Springer. 2003, pp. 381–398.

[35] Hao Lin et al. “Workload-driven VM consolidation in cloud data centers”. In: *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2015, pp. 207–216.

[36] Gueyoung Jung et al. “Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures”. In: *2010 IEEE 30th International Conference on Distributed Computing Systems*. IEEE. 2010, pp. 62–73.

[37] Rodrigo N Calheiros et al. “Workload prediction using ARIMA model and its impact on cloud applications’ QoS”. In: *IEEE transactions on cloud computing* 3.4 (2014), pp. 449–458.

[38] Jingqi Yang et al. “Workload predicting-based automatic scaling in service clouds”. In: *2013 IEEE Sixth International Conference on Cloud Computing*. IEEE. 2013, pp. 810–815.

[39] Peter Bodik et al. “Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters.” In: *HotCloud* 9 (2009), pp. 12–12.

[40] Arijit Khan et al. “Workload characterization and prediction in the cloud: A multiple time series approach”. In: *2012 IEEE Network Operations and Management Symposium*. IEEE. 2012, pp. 1287–1294.

[41] Chunhong Liu et al. “An adaptive prediction approach based on workload pattern discrimination in the cloud”. In: *Journal of Network and Computer Applications* 80 (2017), pp. 35–44.

[42] Nikolas Roman Herbst et al. “Self-adaptive workload classification and forecasting for proactive resource provisioning”. In: *Concurrency and computation: practice and experience* 26.12 (2014), pp. 2053–2078.

[43] Waheed Iqbal et al. “Adaptive prediction models for data center resources utilization estimation”. In: *IEEE Transactions on Network and Service Management* 16.4 (2019), pp. 1681–1693.

[44] Bryan Lim and Stefan Zohren. “Time-series forecasting with deep learning: a survey”. In: *Philosophical Transactions of the Royal Society A* 379.2194 (2021), p. 20200209.

[45] Xiaoyong Tang. “Large-scale computing systems workload prediction using parallel improved LSTM neural network”. In: *IEEE Access* 7 (2019), pp. 40525–40533.

[46] Jing Bi et al. “Deep neural networks for predicting task time series in cloud computing systems”. In: *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*. IEEE. 2019, pp. 86–91.

[47] Hoang Minh Nguyen, Gaurav Kalra, and Daeyoung Kim. “Host load prediction in cloud computing using long short-term memory Encoder–Decoder”. In: *The Journal of Supercomputing* 75.11 (2019), pp. 7592–7605.

[48] Neo Wu et al. “Deep transformer models for time series forecasting: The influenza prevalence case”. In: *arXiv preprint arXiv:2001.08317* (2020).

[49] Shiyang Li et al. “Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting”. In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 5243–5253.

[50] Sifan Wu et al. “Adversarial Sparse Transformer for Time Series Forecasting”. In: (2020).

[51] Ishaan Gulrajani et al. “Improved training of wasserstein gans”. In: *arXiv preprint arXiv:1704.00028* (2017).

[52] Alfred Müller. “Integral probability metrics and their generating classes of functions”. In: *Advances in Applied Probability* (1997), pp. 429–443.

[53] Ian J Goodfellow et al. “Generative adversarial networks”. In: *arXiv preprint arXiv:1406.2661* (2014).

[54] Martin Arjovsky and Léon Bottou. “Towards principled methods for training generative adversarial networks”. In: *arXiv preprint arXiv:1701.04862* (2017).

[55] Cédric Villani. *Optimal transport: old and new*. Vol. 338. Springer, 2009.

[56] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[57] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling”. In: *arXiv preprint arXiv:1803.01271* (2018).

[58] Yao-Hung Hubert Tsai et al. “Transformer Dissection: A Unified Understanding of Transformer’s Attention via the Lens of Kernel”. In: *arXiv preprint arXiv:1908.11775* (2019).

[59] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[60] Bing Xu et al. “Empirical evaluation of rectified activations in convolutional network”. In: *arXiv preprint arXiv:1505.00853* (2015).

[61] Alireza Makhzani et al. “Adversarial autoencoders”. In: *arXiv preprint arXiv:1511.05644* (2015).

[62] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[63] Ashia C Wilson et al. “The marginal value of adaptive gradient methods in machine learning”. In: *arXiv preprint arXiv:1705.08292* (2017).

[64] Erik-Jan van Baaren. “Wikibench: A distributed, wikipedia based web application benchmark”. In: *Master’s thesis, VU University Amsterdam* (2009).

[65] Yanpei Chen et al. “The case for evaluating mapreduce performance using workload suites”. In: *2011 IEEE 19th annual international symposium on modelling, analysis, and simulation of computer and telecommunication systems*. IEEE. 2011, pp. 390–399.

[66] Mohammad Shahrad et al. “Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider”. In: *2020 {USENIX} Annual Technical Conference ({USENIX} {ATC} 20)*. 2020, pp. 205–218.

[67] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In: *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*. USENIX Association, 2004, pp. 137–150.

[68] Michael Ferdman et al. “Clearing the clouds: a study of emerging scale-out workloads on modern hardware”. In: *Acm sigplan notices* 47.4 (2012), pp. 37–48.

[69] Tapti Palit, Yongming Shen, and Michael Ferdman. “Demystifying cloud benchmarking”. In: *2016 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE. 2016, pp. 122–132.

[70] EPFL PARSA. *CloudSuite*. URL: `https://www.cloudsuite.ch` (visited on 06/15/2021).

[71] Matei Zaharia et al. “Apache Spark: a unified engine for big data processing”. In: *Communications of ACM* 59.11 (2016), pp. 56–65.