# Automated Resource Scaling in Kubeflow through Time Series Forecasting

*Adithya Subramaniam
*Deloitte*
*Bengaluru, India*
*adithyasubramaniam27@gmail.com*

Aayush Subramaniam
*Department of ECE PES*
*University Bengaluru,*
*India*
*aayushsubramaniam@pesu.pes.edu*

***Abstract*—With the rapid advent of digitization in the last decade, there is evidence of increased web traffic to applications and, more importantly, load on the underlying technology stack supporting large-scale applications. While failure to address the availability can prove to have a long-term impact, this research study aims to further the auto-scaling of cloud workflows and resources, supporting these applications through advanced forecasting techniques, employing machine learning models like ARIMA, SARIMA, Prophet, and LSTMs to estimate traffic to the application accurately. Large-scale operations are mimicked through Kubeflow, a machine learning platform providing a wide range of capabilities from model development to model inferencing to pipelines, and Prometheus, a time-series database, consuming the forecasted metrics to scale resources and work-flows.**

***Index Terms*—kubeflow, time series forecasting, prometheus, pod autoscaling, kubernetes**

## I. Introduction

Over the last decade, the world has witnessed a significant increase in the number of people accessing the Internet. This digital revolution has been particularly evident in India, Bangladesh, and other South Asian countries. It enables people from even the remotest places to connect with their loved ones over the Internet and interact with their communities. Some reports indicate that India alone has gone from 100 million internet users in 2011 to 800 million in 2023 — an 800% increase in 12 years. However, due to this digitization, as the number of people accessing the Internet continues to grow, it increases the requests and traffic to websites and, more importantly, the load on the underlying technology stack that supports the applications. This massive surge in web traffic can strain infrastructure, causing websites to slow or crash altogether. An example would be the advent of cutting-edge machine learning models, enhancing product experiences through capabilities like search and recommendations. These functions are served through large-scale data pipelines that receive massive scale variety, velocity, and volume of data.

Nevertheless, failure to address the increased traffic would lead to a loss in revenue, particularly impacting large-scale organizations because of the size of their operations. Hence, this underlying infrastructure should be able to withstand the increased traffic, and even more vital for appropriate resources to be available in demand. One possible way to address the challenge of managing resource demand could be to anticipate and prepare for the increase in demand before it occurs. Through this research study, we demonstrate the use of forecasting techniques to predict the anticipated traffic daily and, correspondingly, auto-scaling cloud workloads using machine learning platforms without worrying about human intervention.

## II. Related Works

The literature study aims at understanding contributions made in the field of resource and workload optimization in machine learning platforms, particularly Kubeflow, with the following essays providing greater insight amongst others.

Bioinformatics Application with Kubeflow for Batch Processing in Clouds by David Yu Yuan (B) and Tony Wildish [2] demonstrate the deployment of numerous pipelines in Kubeflow over Kubernetes, leveraging its ideal job scheduling, workflow management, and support for machine learning. Their preference for Kubeflow was evident in the platform's ability to scale and its cloud-agnostic deployment pattern. Concluding their research, the authors emphasized Kubeflow's robust nature to satisfy complex computational bio-informatic pipelines, highlighting the platform's advantages over HPCs in the cloud through its ability to scale and cost-effective GPUs.

The authors in [3] utilized the same dataset, i.e., Google's Web Traffic Time Series Forecasting dataset [1], to develop recurrent neural network sequence-to-sequence (RNN seq2seq) models, furthering the competition's winning model by including the median values that resulted in less sensitivity to the fluctuations in the data. Additionally, the authors leveraged symmetric mean absolute percentage error (SMAPE) to measure performance against existing models, including additional features like window length to validate the model's effectiveness.

[4] discusses frameworks to predict future workloads and provision virtual machines, improving and building upon quality of service (QoS) and latency. They leverage prediction models, including moving average (MA), auto-regression (AR), auto-regression integrated moving average (ARIMA), neural networks (NN), and support vector machine (SVM), evaluating the models using metrics such as Prediction Error,

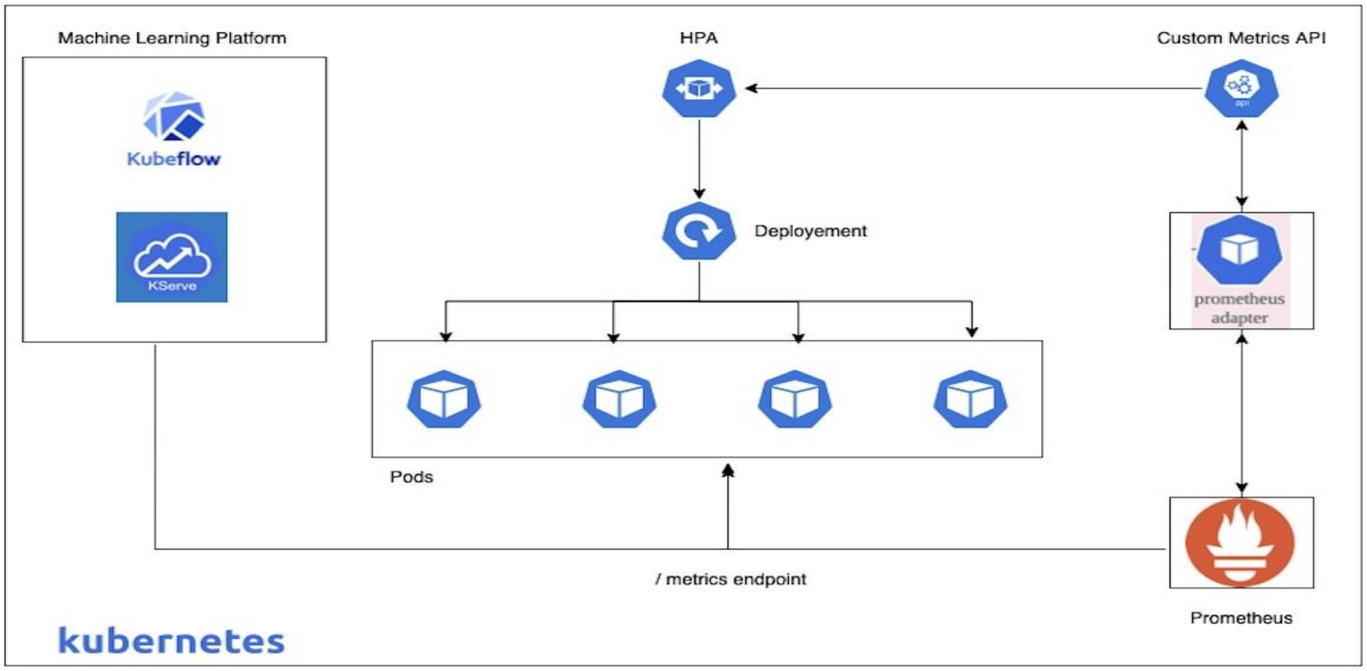


Fig. 1. Architecture Overview

Time Saving, and Under-Prediction and Over-Prediction, with their results indicating that the use of predictive services improves cloud service quality.

Bangyan Du [5] highlights the advanced data processing capabilities of the Apache Spark framework, implementing a distributed large-scale time series analysis system. The system framework is divided into a storage layer, operator layer, and algorithm layer, with the storage layer utilizing HDFS and Hive to organize and index large-scale data and the operator layer implementing operations on Spark for custom time series algorithms.

The research studies listed above and numerous others specializing in time-series forecasting and cloud workload monitoring inspire this study. Further sections highlight using advanced time series models to auto-scale cloud workloads and resources within Kubeflow, extending current capabilities and providing a novel solution to improve scalability in modern machine learning platforms.

## III. Proposed Architecture

The proposed system, as in [Figure 1 Architecture Overview], has three major components - Kubeflow (top left), Horizontal Pod Autoscaler, or HPA (top), and Prometheus (bottom right). The Kubeflow framework enables the storage of all models, pipelines, and model inferences, representing the modern machine learning platform. The metrics of the forecasting models are averaged and inferenced through KServe, a standard inference service built over Kubernetes. The forecasted values are distributed across pods and exposed through the '/metrics' endpoint for consumption by Prometheus.

Prometheus consumes the metrics at particular time intervals, relaying it to the Prometheus Adapter, an interface for Custom Metrics API, further consumed by the HPA. The pod autoscaling mechanism approximates the promotion by which the number of pods is up or down-scaled, depending on the intensity of web traffic to the application.

Sections below further dive into each framework and component in detail.

## IV. Time Series Forecasting

Time Series Forecasting is a statistical technique that analyzes and identifies historical trends and patterns to predict future outcomes. This method has applications in finance, economics, and marketing, where predicting trends is vital to success. Some techniques involve statistical machine learning and other advanced analytical methods.

This study leverages various time series techniques to analyze web traffic data to understand and predict websites' load and allocate resources accordingly. The collected data contains information about the page accessed, the number of visits on a specific date, and the means of accessing the webpage, enabling accurate traffic forecasting by analyzing the current load.

The following sub-section provides a detailed overview of the dataset used to implement this analysis.

### *A. Dataset*

The data source is the Web Traffic Time Series Forecasting Competition hosted by Google, which contains daily web traffic data for over 145,000 Wikipedia articles.

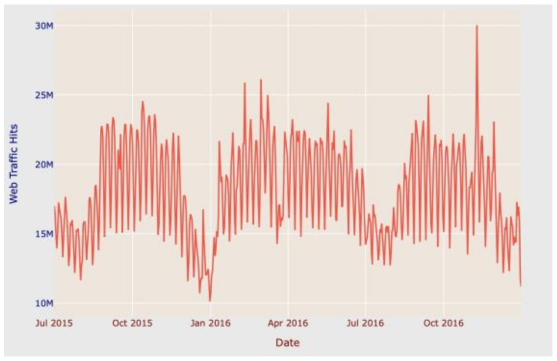


Fig. 2. Visualisation of the Web Traffic Dataset

Each data point contains a 'Page' feature that references the name of the article accessed - formatted as [article-name]*[wikipediadomain]*[accessmethod]. In addition, each value contains the number of times accessed on a particular date, essentially the core attribute that makes our dataset conducive for forecasting.

Other features, not particularly having a strong correlation, include 'type of traffic' - which details the mode of access (mobile, desktop, spiders, et al.)

Constraints during this research study were the need for hourly data, or even by the minute, which would result in a much more insightful forecast, providing traffic metrics on a much more detailed level.

## B. Feature Engineering

Rather than forecasting web traffic for each page, this study aims to forecast it across pages aggregated by language. Different languages include Spanish (es), English (en), French (fr), German (de), Japanese (jp), and Simplified Chinese (zh).

This study emphasizes, in particular, the use of the Spanish dataset, a subset of the larger dataset, because of infrastructure restrictions, with models currently developed and run on Kubernetes clusters with limited memory capacity. Compared to subsets containing other languages, the Spanish dataset incorporates increased variety and variability in web traffic data.

The standard feature engineering process requires the dataset to be indexed by dates, each containing the number of visits to the website aggregated by language.

Further, missing values in the dataset were handled through interpolation, a statistical method that enables the estimation of missing values through related or known values. The parameter 'method' for the interpolating function is set to 'time', making it capable of handling daily and higher resolution data for a given interval.

## C. Forecasting Algorithms and Model Development

To fit the models to historical, time-stamped data in order to predict future values, traditional approaches are applied that include auto-regressive integrated moving average (ARIMA), seasonal-ARIMA (SARIMA), modern machine learning models like Long-Short Term Memory Networks (LSTMs) and decomposable models such as Prophet.

For the best results, the time series dataset must be stationary before fitting a model to the data. Stationary data refers to the dataset that does not have trend, seasonality, cyclical or irregular components in the time series. Additionally, the dataset is said to have constant mean and variance over time.

While the ARIMA, SARIMA, Prophet, and LSTM model

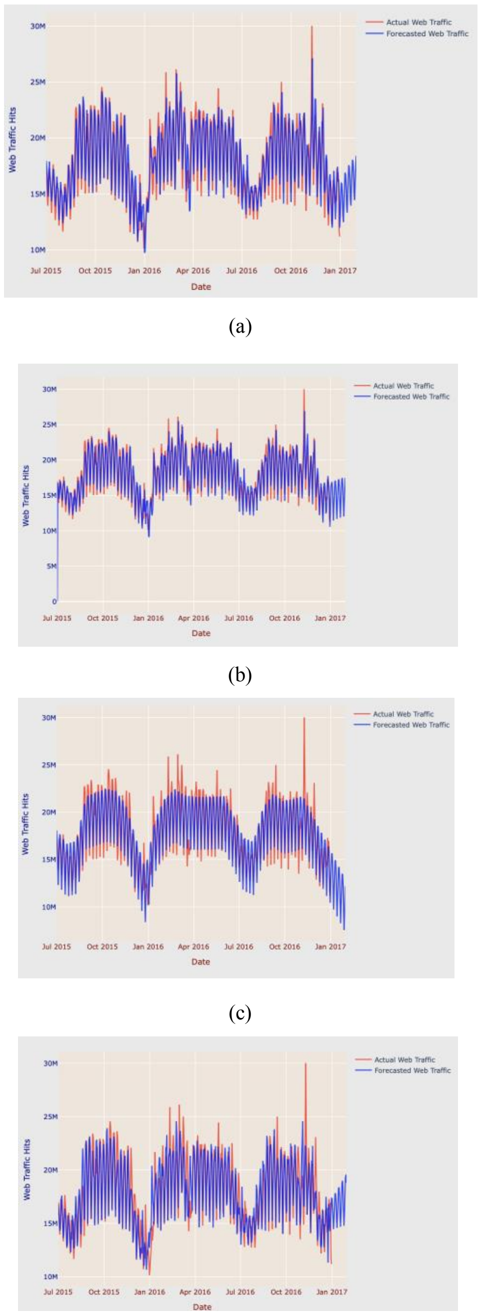


Fig. 3. The Different Time Series Models on the Dataset

architectures do not require data to be stationary, it is ideal to have a stationary dataset to have increased performance. Further, these models aim to capture auto-correlation and lagged dependencies in the data, which assume constant mean and variance. Moreover, reliable and meaningful parameters are estimated when the time series is stationary, helping capture underlying dynamics and generate a more reliable forecast.

If the mean or variance attributes of the dataset change over time, the dataset is considered non-stationary.

This research study utilizes the Augmented Dicky-Fuller Test (ADF Test), one of two standard tests, to identify if the dataset is stationary. The ADF Test is the most popular and most widely used statistical test, done with the assumptions listed below :

- Null Hypothesis (H0): Series is non-stationary
- Alternate Hypothesis (HA): Series is stationary
- p-value greater than 0.05 - Fail to reject (H0)
- p-value less than or equal to 0.05 - Accept (H1)

The dataset used in this research study clears the ADF Test and is stationary, making it conducive for further analysis.

Hyper-parameter tuning : Built-in modules like auto-arima identifies the optimal parameters for an ARIMA, and similar models, by utilizing the commonly-used auto_arima() function, imported from the pd_arima.statsmodels Python library.

Auto-arima conducts differencing tests to determine differencing, d, and then fitting models within defined start(p,q) and max(p,q) ranges.

If the seasonal optional is set to True, auto-arima identifies the optimal P and Q hyper-parameters to determine the optimal seasonal differencing.

The sub-sections below describe each model architecture in detail.

*1)* ***ARIMA****:* Auto-Regressive Moving Averages, or ARIMA for short, is a generalization of the simple autoregressive moving average - ARMA model. These models are used to predict future points in time-series data, with ARIMA utilizing additional regression analysis that indicates the strength of a dependent variable relative to other changing values.

As the name suggests, the ARIMA model consists of smaller components that represent :

- AR - Auto-Regressive, which represents random processes. The output generated linearly depends on previous values, i.e., past observations.
- MA - Moving Average, which represents the linear dependency on current and past observations of a stochastic term.
- I - Integrated, representing the differencing step to generate stationary time series data by removing seasonal and trend components.

Figure 3 (a) depicts a 30-day forecast on the dataset using ARIMA.

*2)* ***SARIMA****:* An extension to the ARIMA model, Seasonal Auto-Regressive Integrated Moving Average or SARIMA, proves to be more robust than ARIMA in the case of highly seasonal time series data. With seasonality in the picture, SARIMA is split into four major components:

- Seasonal Autoregressive Component
- Seasonal Moving Average Component
- Seasonal Integrity Component
- Seasonal Periodicity

SARIMA works best if the dataset has some form of seasonality. From [Figure 2], there are recurring patterns of seasonality. To determine seasonality, the seasonal_decompose function from the statsmodels library comes in handy, identifying weekly seasonality in the dataset.

With the dataset containing seasonal data, particularly weekly, optimizing the model with auto_arima proves to have improved results as depicted in [Figure 3 (b)], a 30-day forecast on the dataset using SARIMA.

*3)* ***Prophet****:* Prophet, developed by Meta, is a framework for time series forecasting based on additive models, where non-linear trends are fit to seasonal data (yearly, weekly, and daily), including holiday effects. Prophet is preferred for time series data with high seasonal effects, as it is robust to unavailable data and shifts in trends, handling outliers well.

[Figure 3 (c)] depicts a 30-day forecast on the dataset using the Prophet framework.

*4)* ***LSTM****:* Long-Short Term Memory, commonly referred to as LSTM, is a Recurrent Neural Network (RNN) that parses inputs in a sequence of features. In the case of sequential data, the function value at one particular step is influenced by the function values at past time steps, pointing to a temporal dependency between such values. LSTMs are particularly adept at identifying temporal dependencies.

The key to the LSTM is the cell state, which enables the flow of information from one cell to the other. The cell represents the LSTMs' memory, which can be updated or forgotten over time. The components that control the updates to the cells are called gates, which regulate the information in cells.

This architecture is beneficial for time series data due to the model's ability to identify long-term sequences of observations, with the research study using a weekly sequence of 7 days and trying to predict the eighth day. Further, the implementation comprises a simple recurrent neural network, using a weekly sequence with one input layer and one stacked layer, and optimizing the number of hidden layers in the network through Ray, detailed in the section below.

[Figure 3 (d)] depicts a 30-day forecast on the dataset using an LSTM model.

**RAY - Distributed Hyperparameter Tuning**

This study utilizes Ray, an open-source framework for scaling AI by providing a compute layer for parallel processing to leverage the computing capabilities of large-scale distributed systems. Scaling machine learning workflows can be achieved through the following:

- Distributing jobs and workloads across multiple nodes and GPUs.
- Configure and access cloud resources.
- Natively leverage and access extensible ML extensions.

Ray is instrumental in tuning hyperparameters of models of different frameworks. For example in particular to this research study, Ray is vital in identifying the optimal values and tuning the hyperparameters of the LSTM model - batch size, epochs, learning rate, and number of hidden layers.

The Prophet and LSTM models, introduced in the previous sections, are tuned through Ray Tune [11], a module within Ray, which follows a three-step process :

- Hyperparameters are tuned in a search space and passed into a trainable object that can be tuned.
- An effective search algorithm is selected to optimize the parameters, and a scheduler, to schedule experiments.
- Together with other configurations, the trainable, search algorithm, and scheduler are passed as parameters into a Tuner to run experiments and create trials. These trials can then be used in analysis to inspect experimental results.

The various error metrics and the general performance of each model will be discussed further in Section VI.

## V. Consuming Published Metrics - Kubeflow, Pod Autoscaler and Prometheus

The Kubeflow [15] project aims to make deployments of machine learning models and workflows on Kubernetes simple, portable, and scalable. There are several benefits to using Kubeflow as an MLOps platform, including :

- Built over Kubernetes, making it easier to run and scale ML workflows on Kubernetes clusters, helping deploy and manage models and underlying infrastructure.
- Includes components like Jupyter Notebooks, Tensorflow Extended (TFX), and PyTorch, enabling experimentation, deployment, and management of models.
- Managing end-to-end machine learning lifecycles, from training to experimentation to deployment, tracking, and management.

Further, specific components of Kubeflow exist that enable the above :

- Jupyter Notebooks - Code Editing
- Katib - Native Hyperparameter tuning
- Model Deployment (Endpoints)
- Kubeflow Pipelines - ML Pipelines on Kubeflow

The core technology that powers Kubeflow is Kubernetes, an automated system for automating deployments, scaling, and managing containerized applications. Kubernetes leverages Pods, which are the smallest computational unit in a cluster. Kubernetes, an orchestration tool, manages pods without external user intervention, brought up through deployments, allowing declarative updates to pods. To enable dynamic IP addresses, the service resource component fields requests from external resources, dispatching them to available pods, enabling users to access applications running on Pods externally.

This paper extends the idea of scaling based on forecasted web traffic to applications hosted on Kubernetes clusters, up or down-scaling the number of pods based on estimated demand. Concerning scaling resources, Kubernetes employs the HorizontalPodAutoscaler, a Kubernetes API Resource, intending to scale the resource workload to match demands. The horizontal pod auto-scaling controller, running within a control plane, periodically adjusts the desired scale to its target to match observed metrics such as CPU Utilisation, Average Memory Utilisation, and other custom metrics specified. Metrics are interfaced through metrics APIs (Resource Metrics API, Custom Metrics API, External Metrics API, et al.).

The HorizontalPodAutoscaler [13] controller operates on the ratio between desired metric value and the current metric value through

$$desiredReplicas = ceil[currentReplicas * (currentMetricValue/desiredMetricValue)]$$

For example, if the current metric value is 100, and the desired value is 50, the replicas in the deployments would be doubled since 100/50 == 2. On the other hand, if the current metric value is 25, the number of replicas in the deployment would be halved.

In current systems, HorizontalPodAutoscaler works on demand while querying resource metrics and auto-scales only on detecting variations during scraping intervals. A major downside to this would be an unprecedented increase in traffic before HorizontalPodAutoscaler estimates the jump, leading to application downtime.

In particular to this research, traffic forecasts would be labeled as custom metrics, configured through Prometheus [14], a time series data store - storing its metrics as time series data. Prometheus interfaces with the custom metrics API through the Prometheus Adapter to utilize these forecasted values for scaling resources.

Prometheus, configured through the '/metrics' endpoint, accesses an application's custom metrics. Prometheus requests metrics through an HTTP GET Request at specific intervals and persists them within its data store. This study utilizes KServe to infer models and store metrics in Prometheus,

TABLE I
METRICS COMPARISION

| Model | RMSE | RMSPE | sMAPE | MAPE |
|---|---|---|---|---|
| ARIMA | 1180303.6416741125 | 0.6613998444215402 | 0.04867339282967528 | 0.04885043236454545 |
| SARIMA | 1391548.724682663 | 0.7896619825927185 | 0.05386275606393117 | 0.051945048247691264 |
| Prophet | 1338200.408069149 | 0.7896957683056572 | 0.05647253550804477 | 0.05688589720660339 |
| LSTM | 1545060.6934851368 | 0.8978423485429876 | 0.06271668884852383 | 0.06317841031724658 |

applying it to HPA. KServe is Kubernetes's standard model inference platform, providing a standard inference protocol across various ML frameworks. Some of its features include :

- Support for modern serverless inference workload with autoscaling.
- High Scalability, density packing, and intelligent routing using ModelMesh.
- Simple and pluggable production serving for production ML serving, including prediction, processing, monitoring, and explainability.
- Advanced deployments with canary rollout, experiments, ensembles, and transformers.

To summarize the different ML models are trained on Kubeflow and served using KServe, mimicking the modern machine learning platform. The applications have a '/metrics' endpoint, in which forecasted daily web traffic value is determined, computing the mean of the models. Following this, the mean persists in Prometheus, which the HPA further extrapolates through the Prometheus Adapter. Finally, the application pods are scaled up or down based on the forecasted traffic value for that day.

The section below will further detail the achieved results.

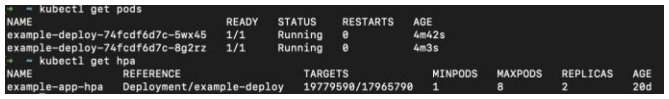
## VI. EXPERIMENTAL RESULTS

Performance metrics are vital components of evaluation frameworks in various fields. Performance metrics in machine learning regression experiments compare the trained model predictions with the dataset's truth. Forecasting has a long history of employing performance metrics to measure how much forecasts deviate from observations to assess quality and choose forecasting methods.

[Table II Mean] draws comparisons between the mean of the truth, i.e., the actual dataset, and the mean of the forecasted results of each model architecture. The results indicate good performance overall, with the Prophet model performing the best.

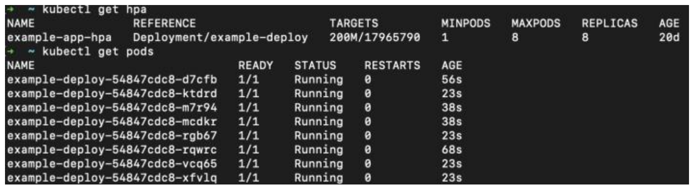
The models are trained on around 500 data points and forecasted 30 days in advance. The metrics of the actual dataset values vs. the model predictions on the same dataset are represented in [Table I Metrics Comparison] compares the most common metrics to evaluate forecasting models.

- **RMSE**: Root Mean Squared Error, or RMSE, depicts the difference between the mean of the absolute values in the dataset to its mean, providing greater insight than the standard deviation of the data.
- **RMSPE**: Root Mean Squared Percentage Error, or RMSPE, is the percentage value of the RMSE of the dataset.
- **MAPE**: Mean absolute percentage error, or MAPE — also called the mean absolute percentage deviation (MAPD) — is an asymmetric measure of the accuracy of a forecast system. It represents the accuracy on a scale of 100, calculating it as the difference between the absolute percent error for each period and truth values over the actual values.
- **sMAPE**: Symmetric Mean Absolute Percentage Error, or sMAPE, is the symmetric mean percentage error difference between the predicted and actual values. [12]



```
→  ~ kubectl get pods
NAME                            READY   STATUS    RESTARTS   AGE
example-deploy-74fcdf6d7c-5wx45   1/1     Running   0          4m42s
example-deploy-74fcdf6d7c-8g2rz   1/1     Running   0          4m3s
→  ~ kubectl get hpa
NAME              REFERENCE                   TARGETS             MINPODS   MAXPODS   REPLICAS   AGE
example-app-hpa   Deployment/example-deploy   19779590/17965790   1         8         2          20d
```

(a)

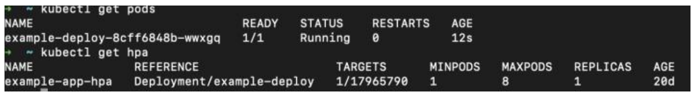


```
→  ~ kubectl get hpa
NAME              REFERENCE                   TARGETS          MINPODS   MAXPODS   REPLICAS   AGE
example-app-hpa   Deployment/example-deploy   200M/17965790    1         8         8          20d
→  ~ kubectl get pods
NAME                             READY   STATUS    RESTARTS   AGE
example-deploy-54847cdc8-d7cfb   1/1     Running   0          56s
example-deploy-54847cdc8-ktdrd   1/1     Running   0          23s
example-deploy-54847cdc8-m7r94   1/1     Running   0          38s
example-deploy-54847cdc8-mcdkr   1/1     Running   0          38s
example-deploy-54847cdc8-rgb67   1/1     Running   0          23s
example-deploy-54847cdc8-rqwrc   1/1     Running   0          68s
example-deploy-54847cdc8-vcq65   1/1     Running   0          23s
example-deploy-54847cdc8-xfvlq   1/1     Running   0          23s
```

(b)

Fig. 4. Pod Scale Up

This study utilizes a Python application image to emulate an application requiring scaled resources, publishing forecasted values to the '/metrics' endpoint. Prometheus leverages the forecasted value and updates the 'gauge' parameter in the client. [Figure 4(a)] depicts the ceil of the mean value of the four different models sent to the '/metrics' endpoint of the application.[Figure 4(b)] illustrates the scenario of increased traffic into the system, preparing the Kubernetes cluster to scale up the pods. Robust models could further estimate outliers, which this research study cannot demonstrate due to computational limitations, similarly, [Figure 5] depicts the scale down.



```
→  ~ kubectl get pods
NAME                           READY   STATUS    RESTARTS   AGE
example-deploy-8cff6848b-wwxgq   1/1     Running   0          12s
→  ~ kubectl get hpa
NAME              REFERENCE                    TARGETS      MINPODS   MAXPODS   REPLICAS   AGE
example-app-hpa   Deployment/example-deploy    1/17965790   1         8         1          20d
```

Fig. 5. Pod Scale Down

TABLE II
MEAN

| Machine Learning Model | Mean |
|---|---|
| Normal Dataset (Truth) | 17965790.54850192 |
| ARIMA | 17978120.600231644 |
| SARIMA | 17908027.540855728 |
| Prophet | 17965644.31604267 |
| LSTM | 17880424.93371961 |

## VII. CONCLUSION

In extending the current implementations of scaling cloud resources and workflows for applications, this research study implements advanced forecasting techniques through different model architectures and designs to estimate the most accurate web-traffic load on applications hosted on modern machine learning platforms, Kubeflow, in this example. To further expand on this study :

- Firstly, this paper can implement different architectures, like the modern transformer, to improve current results.
- Secondly, utilize cloud platforms, such as AWS, GCP, or Azure, to scale infrastructure on the fly.
- Thirdly, collect and utilize larger datasets, preferably by the hour or minute, to achieve an even more accurate and detailed forecast.
- Lastly, data is at the cornerstone of modern machine learning platforms and can be designed to flow back into the system, enabling continual model training.

## REFERENCES

[1] Dataset - https://www.kaggle.com/c/web-traffic-time-series-forecasting

[2] Yuan, D.Y., Wildish, T. (2020). Bioinformatics Application with Kubeflow for Batch Processing in Clouds. In: Jagode, H., Anzt, H., Juckeland, G., Ltaief, H. (eds) High Performance Computing. ISC High Performance 2020. Lecture Notes in Computer Science(), vol 12321. Springer, Cham. https://doi.org/10.1007/978-3-030-59851-824

[3] N. Petluri and E. Al-Masri, ”Web Traffic Prediction of Wikipedia Pages,” 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 2018, pp. 5427-5429, doi: 10.1109/BigData.2018.8622207.

[4] Yazhou Hu, Bo Deng and Fuyang Peng, ”Autoscaling prediction models for cloud resource provisioning,” 2016 2nd IEEE International Conference on Computer and Communications (ICCC), Chengdu, China, 2016, pp. 1364-1369, doi: 10.1109/CompComm.2016.7924927.

[5] B. Du, ”Distributed Large-scale Time-series Data Processing and Analysis System Based on Spark Platform,” 2021 International Conference on Big Data Analysis and Computer Science (BDACS), Kunming, China, 2021, pp. 105-110, doi: 10.1109/BDACS53596.2021.00031.

[6] S. Fischer, K. Katsarou and O. Holschke, ”DeepFlow: Towards Network-Wide Ingress Traffic Prediction Using Machine Learning At Large Scale,” 2020 International Symposium on Networks, Computers and Communications (ISNCC), Montreal, QC, Canada, 2020, pp. 1-8, doi: 10.1109/ISNCC49221.2020.9297301.

[7] S. Bhagavathiperumal and M. Goyal, ”Workload Analysis of Cloud Resources using Time Series and Machine Learning Prediction,” 2019 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE), Melbourne, VIC, Australia, 2019, pp. 1-8, doi: 10.1109/CSDE48274.2019.9162385.

[8] M. Abdullah, W. Iqbal, J. L. Berral, J. Polo and D. Carrera, ”Burst-Aware Predictive Autoscaling for Containerized Microservices,” in IEEE Transactions on Services Computing, vol. 15, no. 3, pp. 1448-1460, 1 May-June 2022, doi: 10.1109/TSC.2020.2995937.

[9] M. N. A. H. Khan, Y. Liu, H. Alipour and S. Singh, ”Modeling the Autoscaling Operations in Cloud with Time Series Data,” 2015 IEEE 34th Symposium on Reliable Distributed Systems Workshop (SRDSW), Montreal, QC, Canada, 2015, pp. 7-12, doi: 10.1109/SRDSW.2015.20.

[10] Lim, Bryan Arık, Sercan Ö. Loeff, Nicolas Pfister, Tomas, 2021. ”Temporal Fusion Transformers for interpretable multi-horizon time series forecasting,”International Journal of Forecasting, Elsevier, vol. 37(4), pages 1748-1764.

[11] @articleliaw2018tune, title=Tune: A Research Platform for Distributed Model Selection and Training, author=Liaw, Richard and Liang, Eric and Nishihara, Robert and Moritz, Philipp and Gonzalez, Joseph E and Stoica, Ion, journal=arXiv preprint arXiv:1807.05118, year=2018

[12] @articleepftoolbox, title = Forecasting day-ahead electricity prices: A review of state-of-the-art algorithms, best practices and an open-access benchmark, journal = Applied Energy, volume = 293, pages = 116983, year = 2021, doi = https://doi.org/10.1016/j.apenergy.2021.116983, author = Jesus Lago and Grzegorz Marcjasz and Bart De Schutter and Rafał Weron

[13] Horizontal Pod Autoscaler - https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale

[14] Prometheus - https://prometheus.io/docs/concepts/metric_types

[15] Kubeflow - https://www.kubeflow.org/docs/started/kubeflow-overview/