Predicting Function Invocations in Cloud Computing

Kaarthik Alagappan
Department of Electrical Engineering and Computer Science
University of Central Florida
Email: kaarthik@knights.ucf.edu

***Abstract*— The world of cloud computing has been growing at a fast rate that most of the business prefer to use function as a service (FaaS) type of usages for their business activities. This type of computing eases the user from having to manage servers themselves and rely on a cloud service provider to execute the uploaded code, thus the cloud service providers are expected robust response times for customer satisfaction. From the cloud service providers' point, a main factor that plays in robust response time is the time to load an application and execute it. A "cold start" in the FaaS means the application has yet to be loaded into the memory for execution and thus will take longer than a "warm start" where the application is already in memory. Knowing when an application will be invoked would be very helpful for the cloud providers in this case as they can expect the execution and "pre-warm" the application. The [1] paper tried to solve this method using a histogram approach coupled with time-series forecasting, and as so this paper tries to expand on the predicting the invocations just during time-series prediction model with the same data. The paper shows how a LSTM-based prediction model would be effective in such situation using real-world data from Azure Functions and how different types of function patterns affect the prediction rate.**

Index Terms **Cloud computing, cold start, LSTM, coefficient of variance.**

I. Introduction

Serverless functions are becoming a hot topic in the cloud computing area due to their ease of access from the users' point of view. The users simply must upload their code to the cloud provider and can invoke it anytime from almost anywhere and the function residing in the cloud will respond. The service interfaces are pretty for anyone to use, and most cloud function services support popular languages like Python and Java, all the more reasons for developers not to look away. It also is a big break for organizations as for once they do not have to dedicate their time to managing a server, nevertheless a cloud server, and can simply rely on the cloud service provider to manage the servers. In addition, users do not have to pay for infrastructure or time that they are not utilizing, so they get charged only for the resources that they use.

Since the functions are available in the cloud, they can be triggered by a lot of different functions. Some examples are timer triggers where functions are invoked periodically based on a set timer, and HTTP triggers where the functions are invoked in response to a HTTP request sent. The users can utilize one or more of these triggers to invoke a cloud function and get a response, trigger other functions, and/or perform other actions in a serverless environment. And when these functions are triggered and invoked, the cloud provider is responsible for loading and executing the triggered function on a server in a robust manner.

Cloud providers aim to speed up the loading and execution time of the function as much as possible for customer satisfaction as it helps the customer have more money and gives them a result fast. The execution time of the function depends a good amount on the code written by the user while the cloud provider can try to optimize the execution process themselves, but the loading time is purely on the cloud server side, thus, the cloud providers aim to reduce the loading time of an application as much as possible. A cold start is when a function is triggered but it isn't already in memory yet so the provider has to load the functions' code and required libraries into memory before it can execute it. And this mostly has to be loaded from a disk most times, thus the read time is also significant. A warm start is when the function is already in memory when it is triggered, thus the cloud provider can skip the loading part and directly execute it. As expected, the cloud providers aim to reduce the number of cold starts to decrease the response time for a function.

The trivial intuition would be to keep the function loaded in memory, that would be infeasible if we are loading all functions into the memory. AWS and Azure have an approach for this called "keep alive" where the function and its resources are kept in memory for about twenty minutes after their execution to avoid quick cold starts [1]. But that solution would be effective if functions are called infrequently, and its resources take up memory for a long time. From the Azure Functions dataset used in this project, it is noticeable that the most frequently trigger is the HTTP trigger and they are not invoked periodically like we would expect from a timer trigger. So, it is inefficient to store a function's resources in memory when the platform is unsure when it would next be invoked.

This project uses a lot of findings from a paper by [1] and tried to expand its work to determine whether it is possible to predict future invocation of functions. The [1] paper mainly focuses on predicting application invocations (application can be a combination of one or more functions) using a histogram method in combination with a time-series predictor to tell the platform when to "pre-warm" an application to avoid cold starts. This project aims to test predicting different types of functions only using a machine learning model with the understanding that if a function is about to be invoked, the application to which the function belongs to can be marked as to be pre-warmed.

II. Related Work

A. Serverless in the Wild

As stated previously, this project draws heavy inspiration from the paper by Shahrad et at., (2020) [1] where the authors use the same dataset from AzurePublicDataset repository [2] and propose a method to predict the invocation of a function so that the cloud service provider can anticipate an invocation and pre-load the application in memory.

Their proposed method for solving this is to use a histogram method consisting of the idle-times to identify pre-warm and keep-alive windows for an application using which the cloud service provider can infer when to expect an application wide invocation and load the application into memory and how long to keep the application in memory as opposed to a fixed time in current implementations. When enough data is available and a pattern can be found, it is stated that the histogram provides a good insight into the pre-warm and keep-alive windows for an application, but when not enough data is available or when the pattern is not consistent, the authors switch back to a standard keep alive policy till the algorithm can learn a pattern. And when the histogram is small due to the limit data or pattern, then the authors use a time-series based prediction function.

Their test results show a reduction of 15% cold starts compared to previous Azure policies. Since the authors are from Microsoft and they worked with their own data from Azure Functions, the authors got to implement their findings in the Azure system itself where they say it is currently being used to reduce the number of colds starts.

B. A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment

The paper [3] uses a different LSTM approach to predict workload in a cloud environment. They propose to use the LSTM with an attention mechanism that adds weights to historical data to ensure that only relevant historical data is used to predict the future workload. The LSTM model that they propose includes an encoder and a decoder, both of which are independent of each other. The encoder takes in the time sequence-based data traces and transforms it into a context vector that the decoder takes in and generates intermediate prediction values, which in return is used by the final output layer to make the prediction.

The attention mechanism is implemented in the LSTM model where each historical data is assigned a weight based on how important that data in predicting the future workloads, and is updated periodically. And they have a custom output layer where it has three-layers. Two of which have a parametric rectifier linear unit and third layer is a sigmoid function to get an output between 0 and 1. This type of method takes a way different approach than this paper and [1] in that they are only considering part of the historical data based on the weights, not all of them and they use a custom output layer with three different activation functions. They also mention that their model performs better than the ARIMA model that [1] uses.

Trigger	%Functions	%Invocations
HTTP	55.0	35.9
Queue	15.2	33.5
Event	2.2	24.7
Orchestration	6.9	2.3
Timer	15.6	2.0
Storage	2.8	0.7
Others	2.2	1.0

Fig. 1. Statistics on Functions and Invocations triggers. From [1]

C. Predicting Cloud Resource Utilization

It is hard to find a paper other than the previously mentioned one that directly compares to a invocation-prediction solution for resource utilization. The paper [4] is another paper in this field that aims to predict the resources that will be needed for a future task in cloud platforms. Our proposed solution and [1] take on the approach of figuring out cloud resource allocation was to predict what function or task will be invoked in the future, but this paper takes on the next step and aims to predict what resources will be needed for a given task.

They also use a time-series based artificial neural network that they designed for the paper that takes in what type of task is incoming, input data for that task, and what resources are available. The artificial neural network takes in this input, predicts what resources will be needed for the given task and its input, and finally after task execution the predictor takes in the actual resources used for that task and input and used it to make better prediction in the future. Their proposed approach isn't exactly similar to a time-series predictor but scaled up to it.

III. Dataset

As mentioned before, this project is using the publicly available dataset for Azure Functions [2]. The dataset consists of invocation, memory, and duration data for more than 65,000 Azure functions for more than 20,000 different applications. In this case a single application can consist of more than one function. This project solely focuses on predicting next set of invocations for a specific function, so it focuses on the dataset consisting of the invocations for each function.

The invocation data consists of data for every single minute in a fourteen-day range. So, for a given function there is about 1440 data points available per day stating how many invocations happened per minute in that day. The [1] paper does a detailed evaluation of this dataset that was used to guide this project. As they reported in Figure 1, HTTP triggers are the most common triggers amongst the functions by a big margin, so it would be good to focus on them specifically since they represent a major amount of the functions and invocations in the dataset. Following that is the queue, event, orchestrator, and timer trigger types in the order of popularity.

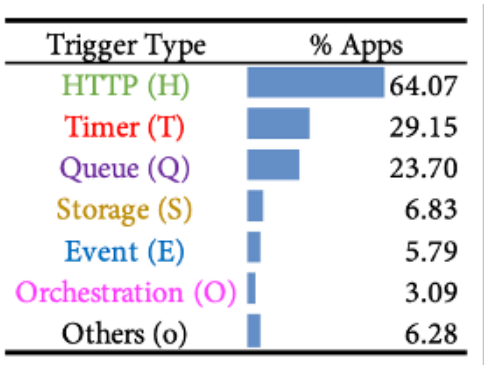

Fig. 2. : Statistics on Triggers Per Application. From [1]

But if you look at Figure 1, we can see that at least 90% of the applications either have a HTTP trigger or a timer trigger, showing that they are the primary triggers present in Azure Function applications. So, this paper focuses on how prediction those two triggers occur at different stages (explained shortly) since they two are the most used triggers in applications. The paper [1] tries to predict the application invocation pattern as only applications are loaded into memory when a function is triggered, so they take into account a holistic view of the application with one or more functions to do their histogram-based prediction. This paper is modeled on the fact that it should be sufficient to predict the invocation pattern of just functions as the Azure platform should know which function belongs to which application, as such with that information if a function is predicted to be invoked in the near future, the platform can take that input and pre-warm the application that the function belongs to. One might expect that since timer functions are triggered using a timer, they are done in a periodic fashion and hence are easily predictable. But the datasets tell a different story.

The inter-arrival time for function is the time between invocations of a function, it tells us how frequently or infrequently a function is invoked and gives us a glimpse at how predictable a function is in our case. The coefficient of variation of inter-arrival times, which is the standard deviation of the inter-arrival time of a function divided by the mean of the inter-arrival time, gives us a definite number indicating the variation in invocation pattern for a function. A coefficient of variation (CV) of 0 indicates that the function is triggered in a very consistent manner and has no outlier invocations. A CV greater than 0 indicates that there is variance in the invocation pattern and thus ends up being a bit challenging to predict. The authors of [1] state that human generated triggers tend to follow a Poisson pattern and are expected to have a CV of 1, and a CV greater than 1 are very challenging to predict.

Based on this, a CV of 0 is expected for timer triggers as we could imagine that they are invoked in a consistent pattern based on some set time. But doing the calculation and categorizing the triggers with the respective CVs, only around 25% of the extracted timer trigger functions from seven days' worth of data have a CV of less 0.05 while interestingly around 26% of the HTTP trigger functions have a CV less than 0.05. And from an application standpoint, only around 50% of the applications with just timer triggers have a CV 0 [1]. With this we can say that not all timer trigger functions are easily to predict while some other non-timer triggers like HTTP are also invoked periodically and could be predictable. Given that, the project tests out how effective a LSTM-based predictor model is with a timer and HTTP triggered functions with varying CV values to get an idea on how easily or challenging it is to prediction with a machine learning model.

IV. Methodolgy

We want to build a machine learning model that would take in account the previous history of the function invocations and use that data to predict the future invocations of that function. This is called a time-series forecasting model and the long-short term memory (LSTM) model is a well-known model for the time-series forecasting problems. LSTM is a recurrent neural network that specializes in predicting a value based on previous history, hence, it is a good fit for this problem.

A. Data Processing

We are going to be predicting values for a single function at a time, so our data is going to be a cumulation of all available times for a chosen function. If a function's data is available for all fourteen days, then that dataset will have about 20,160 data points available. One thing to be noted that is that some functions are not present in all the fourteen days the data has been collected, for which a reason could be that the function was added or removed from an application during the fourteen-day period. The invocation values of the chosen function from all fourteen data files are collected and compiled into a single CSV file with one column starting from time 0 of the first day.

As much as we wanted to test out a majority of the functions in our machine learning model, we could only test out a few due to computational limits. To narrow down what functions to test out in the model, functions of HTTP or timer triggers were only retained because they were the most used functions amongst all applications (based on Figure 1). Since we can expect a function with a CV of 0 to be relatively predictable, we also wanted to test out how the machine learning model performs with functions whose CV values are greater than 0. So, from timer and HTTP trigger functions, we sort each function based on their CV values. And amongst those, we choose one function for each trigger type (timer and HTTP) for each of the following CV values: 0, 0.25, 0.5, and 1. Those values are chosen because their results will give us a good view on how the model performs when predicting functions where the invocation patterns of the functions are getting more and more diverse. We also enforce a threshold for choosing functions where we avoid functions that have less than twenty five invocations to ignore functions with less number of values, functions with a IAT standard deviations less than 5 or have a IAT mean of less than 1 since we want to have a diverse set of invocations to test the model under different settings.

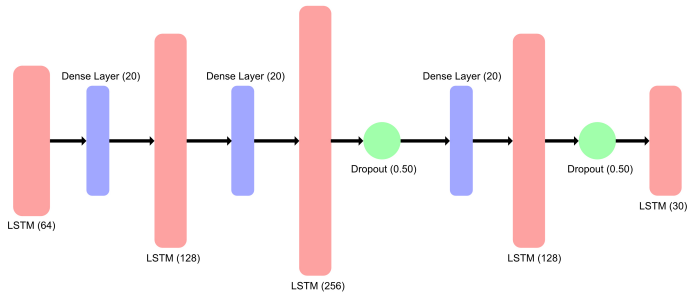

Fig. 3. LSTM Architecture

With the 8 chosen functions, each function's data was split by a 80-20 ratio where 80% of the data was used to train the model while 20% was used as test data. The model was built to use 3 hours of historical data to predict the next 1 hour of data. This window was chosen through a few runs with different historical data windows and this seemed to produce good results with good loss values for most functions.

B. LSTM Model

The LSTM model was chosen for this problem as it is a well-known time-series prediction algorithm and seemed to be a perfect fit for this problem. And mainly because I wasn't able to find a good number of tutorials to understand other types of time-series forecasting models. The tutorial at [5] was used to build a basic version of the LSTM model through which the datasets were initially trained with. Most of those models had some problems because they couldn't handle a lot of data coming in. So, a few more LSTM layers were inserted along with dense and dropout layers to avoid over-fitting problems. The architecture of this model is show in WHATFIG. We gradually increase the LSTM layer size and decrease it towards the end for a better training process, and the dropout layers are used for regularization in order to avoid over-fitting the model because of increased number of parameters. The model was trained with 150 epochs and 100 steps at each epoch, and it used the Adam optimizer and MSE regression loss function.

V. Results

The proposed architecture was built using TensorFlow and was run on a MacBook Pro model without GPUs and Google Colab (Google Colab had a restriction for how many minutes we can use their GPUs, hence some experiments were also run locally). The model was trained ten times for the corresponding ten training data files, two fully traced function data, one a timer triggered function and one a HTTP triggered function, for each of 5 chosen CV rate. It is expected that as the CV increases for a function the accuracy rate of the model would decrease as if the CV were high it means that there is not a definite pattern for that function. The loss values and accuracy values for each training run are shown below. The output values from the LSTM model are not perfect integers, rather as decimal number values. So for our case the numbers are rounded up or down based on the predicted number since there can be decimal values for number of invocations at a given time.

A. CV of 0

The following are the accuracy and loss values of training the model with a timer and HTTP function data with a CV 0

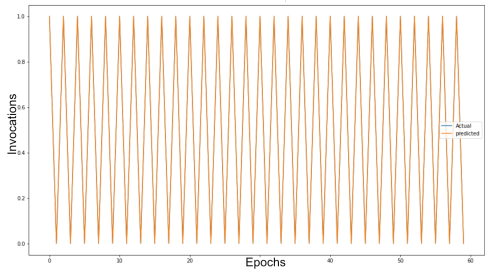

Fig. 4. 0 CV Timer Function Actual vs Predicted Values

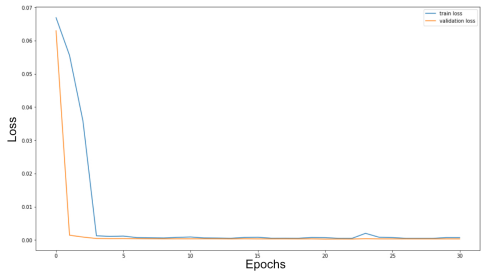

Fig. 5. 0 CV Timer Function Validation and Testing Loss

The model predicted the pattern of function in this case and managed to predict all the invocations at the right times. It has a good validation and training loss value progression as well.

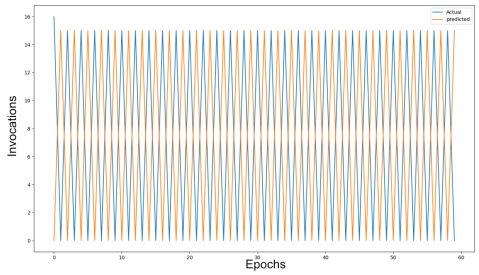

Fig. 6. 0 CV HTTP Function Actual vs Predicted Values

Though not as perfect as with predicting the previous timer function, the model still managed to find the pattern of the invocations and predicted them 1-2 minutes early, which can still be useful for a cloud platform for a prediction.

B. CV of 0.25

The following are the accuracy and loss values of training the model with a timer and HTTP function data with a CV 0.25

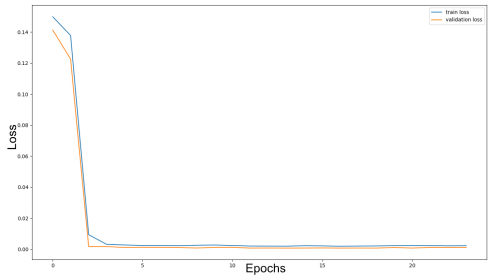

Fig. 7. 0 CV HTTP Function Validation and Testing Loss

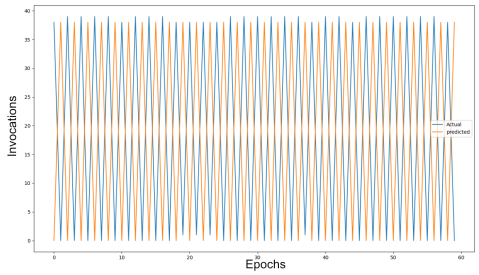

Fig. 8. 0.25 CV Timer Function Actual vs Predicted Values

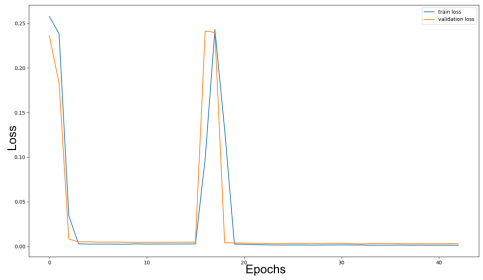

Fig. 9. 0.25 CV Timer Function Validation and Testing Loss

When there is a bit more noise in the invocation pattern, the model still managed to predict the pattern but as with Figure 6 it is predicted the values 1-2 minutes before the actual invocation. Looking at the loss values, we can see that there is a pattern of over-fitting as the validation loss is pretty high compared to the training loss. This could be because the training dataset contains a less amount of varying IAT values and the model was just memorizing that pattern and couldn't generalize it.

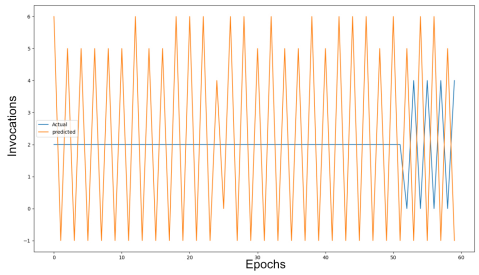

Fig. 10. 0.25 CV HTTP Function Actual vs Predicted Values

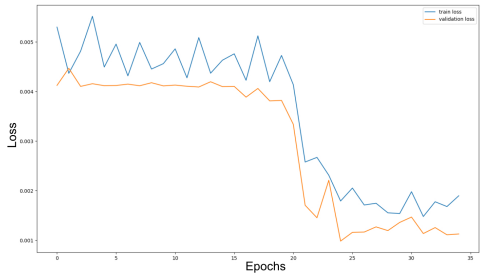

Fig. 11. 0.25 CV HTTP Function Validation and Testing Loss

With the HTTP function it gets a bit trickier with the varying values and as you can see in Figure 10 it is not predicting the values correctly at first but gets better towards the end of the prediction window, which could indicate that it learned a pattern but the testing values different from that pattern a bit. But unlike the timer function the accuracy loss values and validation loss values are on par with each other so no there isn't over-fitting or under-fitting.

C. CV of 0.5

The following are the accuracy and loss values of training the model with a timer and HTTP function data with a CV 0.5

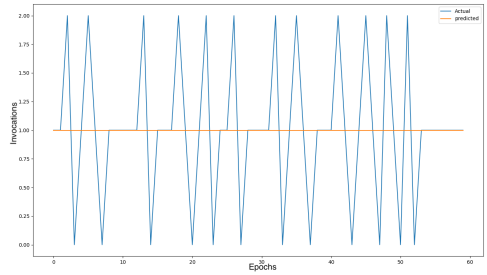

Fig. 12. 0.5 CV Timer Function Actual vs Predicted Values

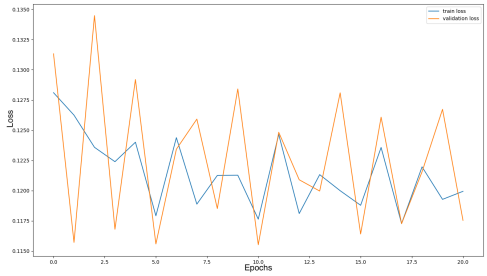

Fig. 13. 0.5 CV Timer Function Validation and Testing Loss

And with a CV of 0.5, the model predicted some correct values for the timer function at few places, but most of the predicted values are wrong. And the model did not show a pattern of over-fitting or under-fitting as you can see from Figure 13 but did have relatively high loss values. And with the HTTP function the values are completely off. As you can see in Figure 14 the actual values have a high invocation variance and model

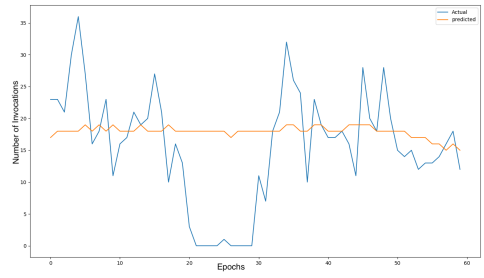

Fig. 14. 0.5 CV HTTP Function Actual vs Predicted Values

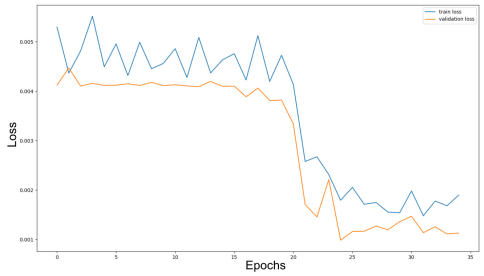

Fig. 15. 0.5 CV HTTP Function Validation and Testing Loss

predicted value does have more variance than other models but still had a high error margin. As such the loss values do indicate bit of over-fitting issue here as well.

D. CV of 1

The following are the accuracy and loss values of training the model with a timer and HTTP function data with a CV 1

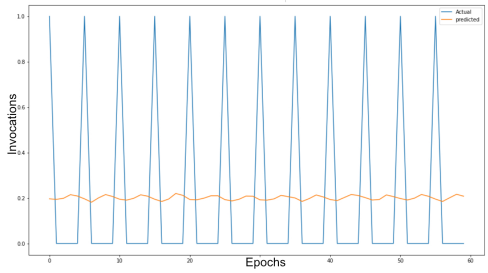

Fig. 16. 1 CV Timer Function Actual vs Predicted Values

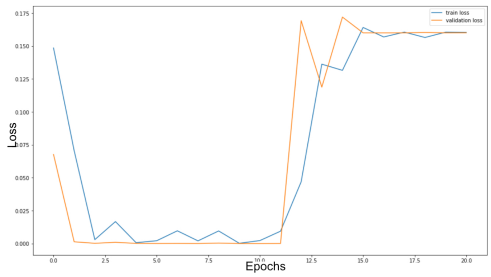

Fig. 17. 1 CV Timer Function Validation and Testing Loss

With a function with CV 1 we are not expect good predictions as a CV of 1 indicates that the pattern of invocation is not consistent at most points. Also something to note here is not that I am not rounding the values predicted in this model to show that the model learned the pattern but did not predict the correct values in this case, which is something that could be worth looking into (whether learning the pattern is good enough). The model did have a high jump in the loss values as seen in Figure 17. This could be because of the initial weights randomly assigned. The predictions for the HTTP also have a

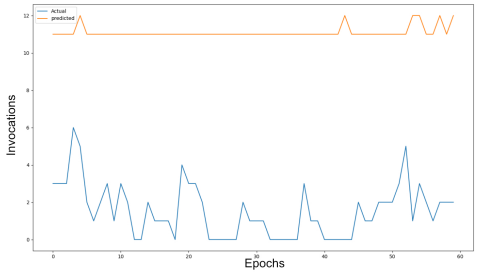

Fig. 18. 1 CV HTTP Function Actual vs Predicted Values

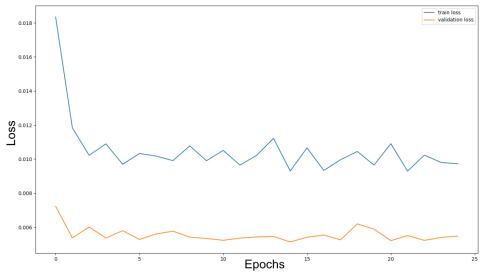

Fig. 19. 1 CV HTTP Function Validation and Testing Loss

high margin of error and the model seems to be overfitting here as well.

1) Predicting Functions with Application Wide Data: The following section provides some insight into out the LSTM model would perform if we try to predict a function with the invocation history of that function and all other functions part of the application it belongs to. The dataset does not mention if certain functions are related with each other in an application context, so this model was trained with an application data to test out how the model performs.

The trained application had five functions in it, 4 timer triggered functions and 1 HTTP triggered functions. The predicted value is one of the timer functions whose CV was 4 and it used the past 3 hours of historical data of all the functions to predict the next 1 hour of invocations of the specific timer function.

The model surprisingly does well for a timer triggered function whose CV was 4, and upon inspecting the specific function it seems like that function has a pretty good invocation rate but there some relatively big breaks in between that caused the CV to go high. Running that specific function through the model

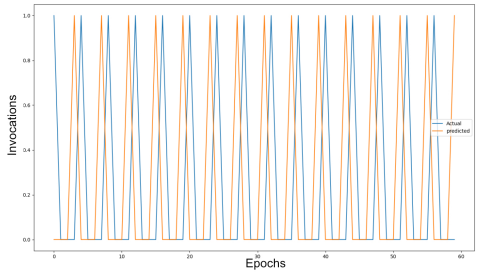

Fig. 20. Application-wide Actual vs Predicted Values

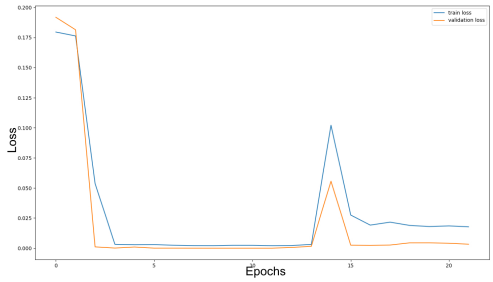

Fig. 21. Application-wise Validation and Testing Loss

individually yields the a perfect accuracy value as seen in Figure 22. Based on this we can hypothesize that if the function has

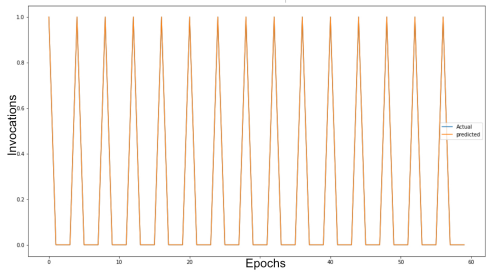

Fig. 22. 4 CV Timer Function Accuracy Values

a good inter-arrival time for most of its lifetime, the model should still be good to predict its values even if there are big discrepancy in the IAT but in small numbers.

VI. Conclusion

In this paper, a LSTM model was built to predict the future invocations of specific functions present in the Azure Functions service. A real world dataset was used for training and testing purposes and specific functions were narrowed down based on certain criteria to have good variety in the training datasets. The functions were mainly of timer and HTTP trigger based as those two were the most common triggers associated with functions used in applications and the chosen functions had a inter-arrival time coefficient of variance rate ranging from 0-1 to test the model when there is little, moderate, and high variance in the invocation patterns.

Based on the results, we can say that as the CV increases the accuracy of the model decreases because of the unpredictable invocation patterns. The model did relatively well for a CV of 0.25 compare to CV of 0.5 and 1, but it performs the best when the CV is around 0 or when the number of places the invocation IAT increases is very slow. So the model is tolerant to certain degree of varying patterns and can be useful for predicting when those specific set of functions will be invoked.

I wanted to test more functions of different variety and test the model based on applications as well and see how well it holds up. It could be beneficial to look into a loss function that is specific for this problem as the proposed model was using the mean squared error loss function, which is known to be a good regression loss function, but designing a custom loss function could improve the results and a possible future direction for this project. The code for the proposed algorithm is available at https://github.com/kaarthikalagappan/predicting_azure_functions_lstm

References

[1] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich, and R. Bianchini, "Serverless in the wild: Characterizing and optimizing the serverless workload at a large cloud provider," 03 2020.

[2] "Azurepublicdataset/azurefunctionsdataset2019.md at master · azure/azurepublicdataset." [Online]. Available: github.com/Azure/AzurePublicDataset/blob/master/AzureFunctionsDataset2019.md

[3] Y. Zhu, W. Zhang, Y. Chen, and H. Gao, "A novel approach to workload prediction using attention-based LSTM encoder-decoder network in cloud environment," *EURASIP Journal on Wireless Communications and Networking*, vol. 2019, no. 1, p. 274, Dec. 2019.

[4] M. Borkowski, S. Schulte, and C. Hochreiner, "Predicting cloud resource utilization." New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2996890.2996907

[5] V. Lendave, "How to do multivariate time series forecasting using lstm," Jul 2021. [Online]. Available: https://analyticsindiamag.com/how-to-do-multivariate-time-series-forecasting-using-lstm/