MLaaS: A Framework for Exposing Machine Learning as a Service on Cloud Platforms

Sandesh Gade, Abhimanyu Banerjee, Gautam Somappa

Abstract: Machine Learning has been seeping into every sphere of our lives from autonomous driving, movie recommendations and online shopping to targeted advertising campaigns, detecting anomalous masses in the brain and stock market analysis. With the impact machine learning has had on our lives, there have been advocates for democratizing the science behind it to make it more accessible to people who might need it. In our project, we aim to make a contribution to this drive by exposing the power of machine learning as a service on a cloud based distributed platform. We have striven to abstract the intricacies of training and predicting on data as much as possible to allow the end user to focus on the application of the *machine learning* rather than the science behind it. We have designed a distributed framework capable of both automatic scale-up and scale-out to ensure that the user gets the performance they seek from the service without any hiccups.

1. Introduction:

Machine Learning[1] has boomed in the last few years from being used by just a handful of computer engineers exploring whether computers could learn to play games and mimic the human brain, to a broad discipline that has produced fundamental statistical and computational theories of learning trends and patterns in any and all forms of data. In recent years, machine learning has been incorporated into almost all applications, from driving cars to monitoring your heart-rate in order to detect anomalies. But even now, *machine learning* is still inaccessible outside a niche community pushing forward it development. However, that trend is changing and in recent years, we have seen the community working towards making machine learning accessible to all [2].

Our project aims to give anybody access to the learning capabilities of machine learning without the baggage of having to understand the inner workings of its various algorithms. Additionally, the power of machine learning is amplified by the benefits of cloud computing. Users can offload the computational and management heavy-lifting to the cloud wherein the performance guarantee is ensured by the inherent automatic scale-out and scale-up design principles of the same. Our framework is primarily targeted towards IoT developers in need of real-time training and prediction on raw data, which can be accomplished through our API endpoints. However, we have designed our framework to also allow for a more hands-on and guided setting up of a machine learning pipeline through an intuitive user interface.

2. Related Work:

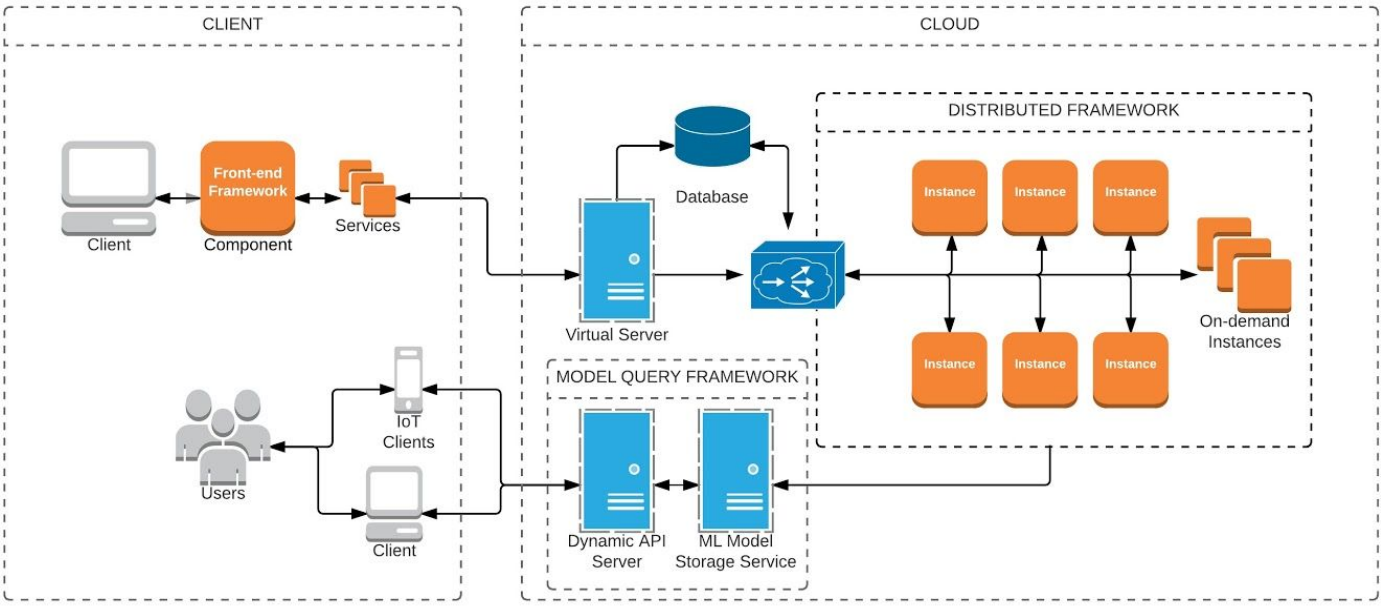
Machine Learning as a Service: Baldominos et al.[3] also proposed a platform built on top of Hadoop. Its implementation was capable of handling up to 30 requests at one time while maintaining a response time of less than one second. Our implementation, on the other hand, is

not built on top of any existing distributed computing framework. OpenCPU is another open-source platform, launched in 2014, that creates a Web API for R, a popular statistical analysis software environment. However, because it is practically a middleware for accessing R functions, it does not take into account many non-functional requirements like scalability and performance.
In the industry, Google, Microsoft, and Amazon have been releasing their own proprietary platforms. Google released its Prediction API2 [4] in 2014 but it largely allows for prediction capabilities on Google's own pretrained machine learning model. Also in 2014, Microsoft launched Azure Machine Learning .and inn 2015, Amazon released AWS Machine Learning. Their popularity proves that the demand exists but unfortunately, the designs and implementation specifications of these products are not publicly available.
PredictionIO, OpenCPU, and Baldominos' platforms are built on top of a specific analytical tools and suffer from its restrictions. This means less flexibility for adding new machine learning algorithms, for data storage, and for deployment. Although Hadoop and R are open-source projects, it is not a trivial challenge to adapt them to a new approach. The same happens with the industry players and their proprietary solutions when external developers cannot have access to the code to add new algorithms.

Machine Learning Through An Interface: We have striven to abstract the complexities involved with setting up a machine learning pipeline as much as possible. This allows users to focus on the results rather than the intermediate steps of reaching their goals. The closest to our work is Keras [5] which offers an easy to use programming interface over more involved backend libraries like Tensorflow [6] or Theano [7]. Our implementation on the other hand is library agnostic and in our attempt to provide fast and reliable results, we have chosen whichever library or tool is best suited to the task at hand.
Moreover, different models can perform better or worse, depending on the used algorithms, parameters and data set. There is no such a thing as the best learning algorithm. For any algorithm, there are data sets that perform very accurately and others that perform very poorly. For the same data set, different algorithms can perform differently because of their own nature. MLaaS helps the user to run multiple algorithms and compare their performances, so the most suitable algorithm can be chosen.

3. Implementation Details:

Our approach for this project is to build an open source cloud framework powered by a highly custom server-client architecture that handles user interaction with management of machine learning tools.



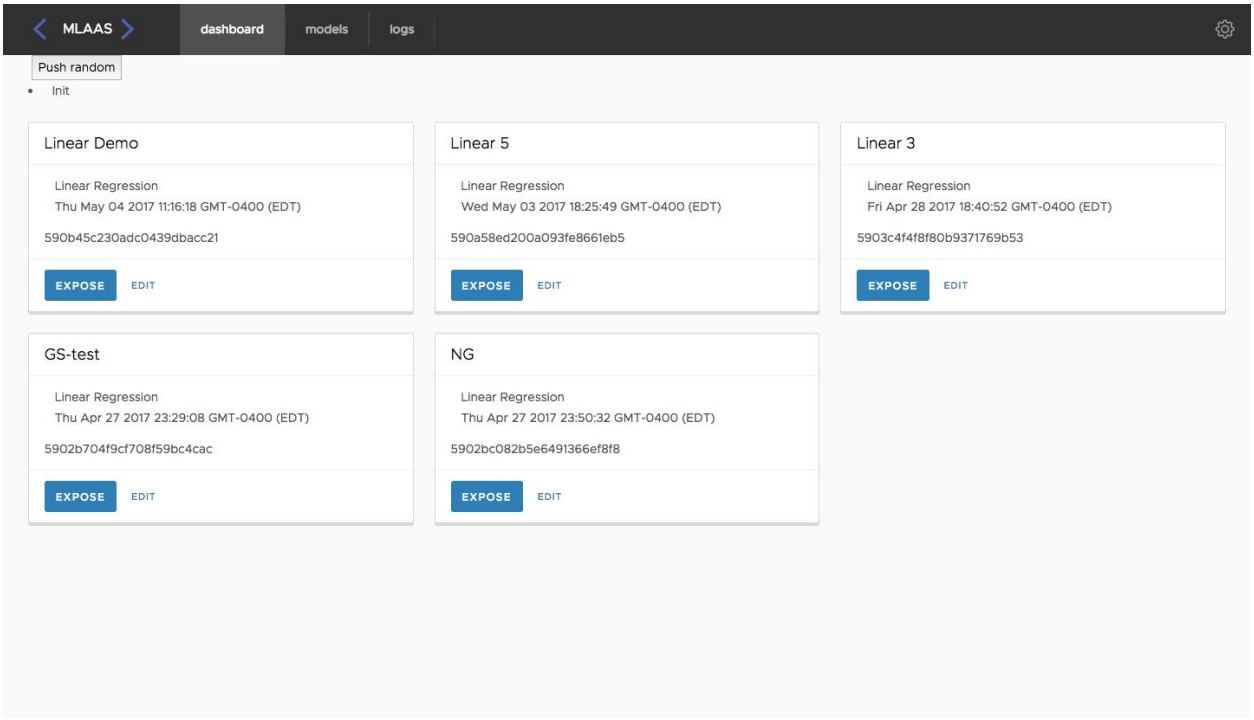
The architecture of our proposed model is described above. The architecture is separated into two primary zones - the Client and the Cloud. The MLaaS framework ties components in both these zones together and utilizes the developed mechanisms to alleviate a seamless interaction between the user and cloud framework. The user interacts with a client application which allows several interactions to access the functionalities and features of the MLaaS cloud framework. The user can create a machine learning model using the front-end client application which simultaneously initializes a shared placeholder that has all the information related to the model to be trained including, but not restricted to, the model name, type, parameters and the dataset to be trained. The different components and its role in the architecture is described below:

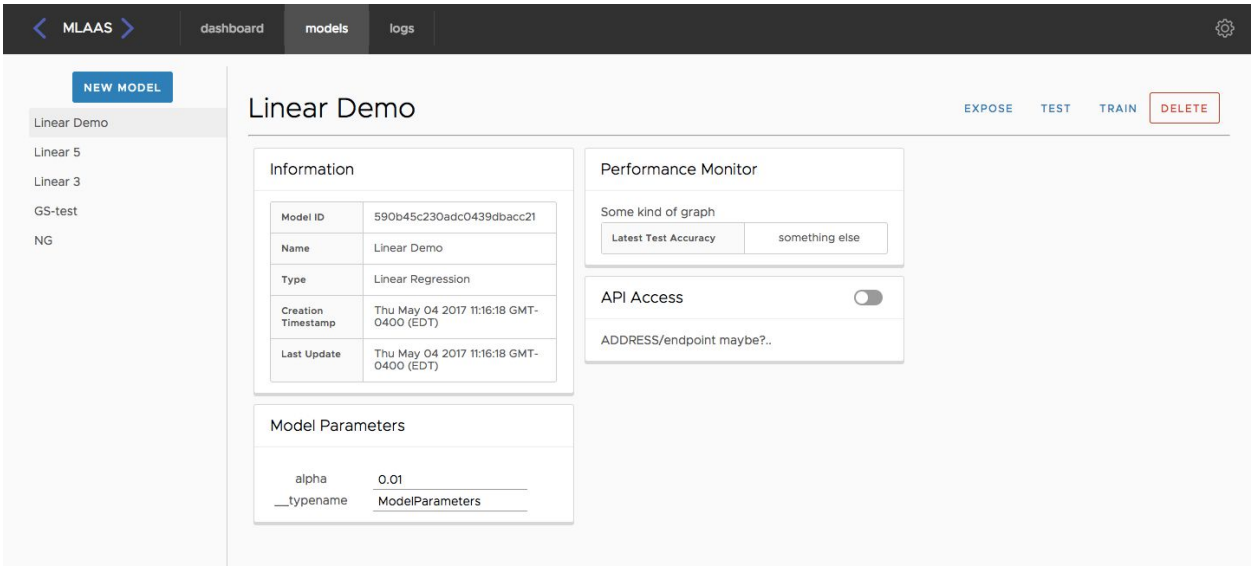
1. Client:
    a. The client application allows the user to manage multiple machine learning models and offers interfaces for these models to be exposed through the cloud as a personalized RESTful service.

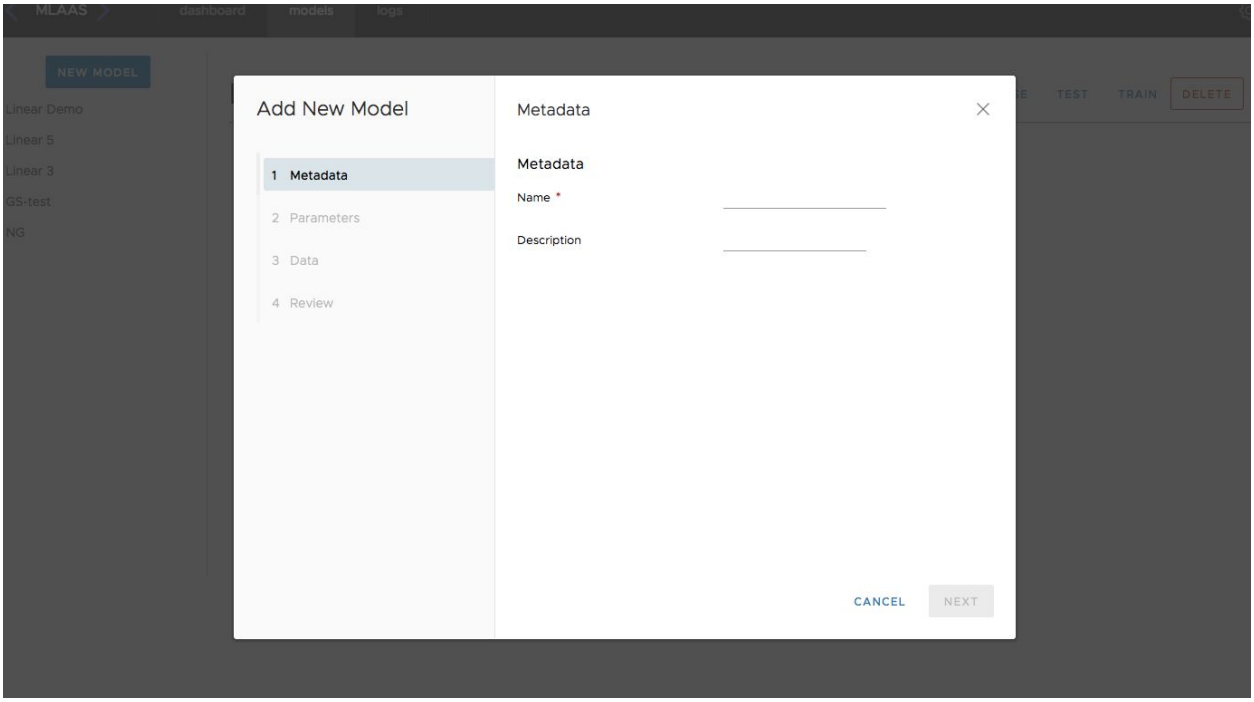
    b. The client application exposes an interface to the user for the management of machine learning model. It is responsible for assisting the user with the following operations:
        1. Creation of a Machine Learning Model
        2. Reading Properties of an existing Machine Learning Model
        3. Updating the Properties of an existing Machine Learning Model
        4. Deleting an existing Machine Learning Model
        5. Triggering the training of an existing Machine Learning Model on the Distributed Framework.

6. Displaying the URL for the user to access the Machine Learning model for real-time use.

c. The following screenshots highlight the main features of the Client Application







2. Cloud:
    a. Server Application:
        i. The server application is responsible for serving connected users with the client application and handling requests for creation, fetching, updating and deleting machine learning model along with user data.
        ii. It interacts with a shared database instance to store the state of each machine learning model that is created by the user. This following screenshot displays the number of connections and the logical size growth of the database over a duration of the past two months. The current configuration for the database maintained 3 shards of the replica set of the primary database. This highlights an important performance criteria for high-usage database systems. By maintaining multiple replicas of the database, a database system is able to cater to multiple incoming and outgoing requests/queries for data. Since our database was shared between the server application as well as the worker nodes in our distributed framework it was critical to provide a cloud database instance that was highly available.

iii. It also forwards task-specific requests to appropriate endpoints in the architecture.

b. Distributed Framework:

i. Certain task-specific actions such as training and testing machine learning models require significant computational and storage resources. Requests for such tasks are routed to a distributed framework that is able to provision nodes for the computation tasks on the availability of resources to carry out the task.

The distributed framework maintains multiple queues for different tasks. This allows for a separation of concerns implementation for not overloading a cluster of worker. Operations that require training and testing machine learning models are locally executed on the worker nodes of the cluster independent from other nodes. This allows the cloud framework to easily scale up and scale out as and when required depending on the load on the distribution manager. The screenshot below highlights a section of the container log from the distributed framework services.

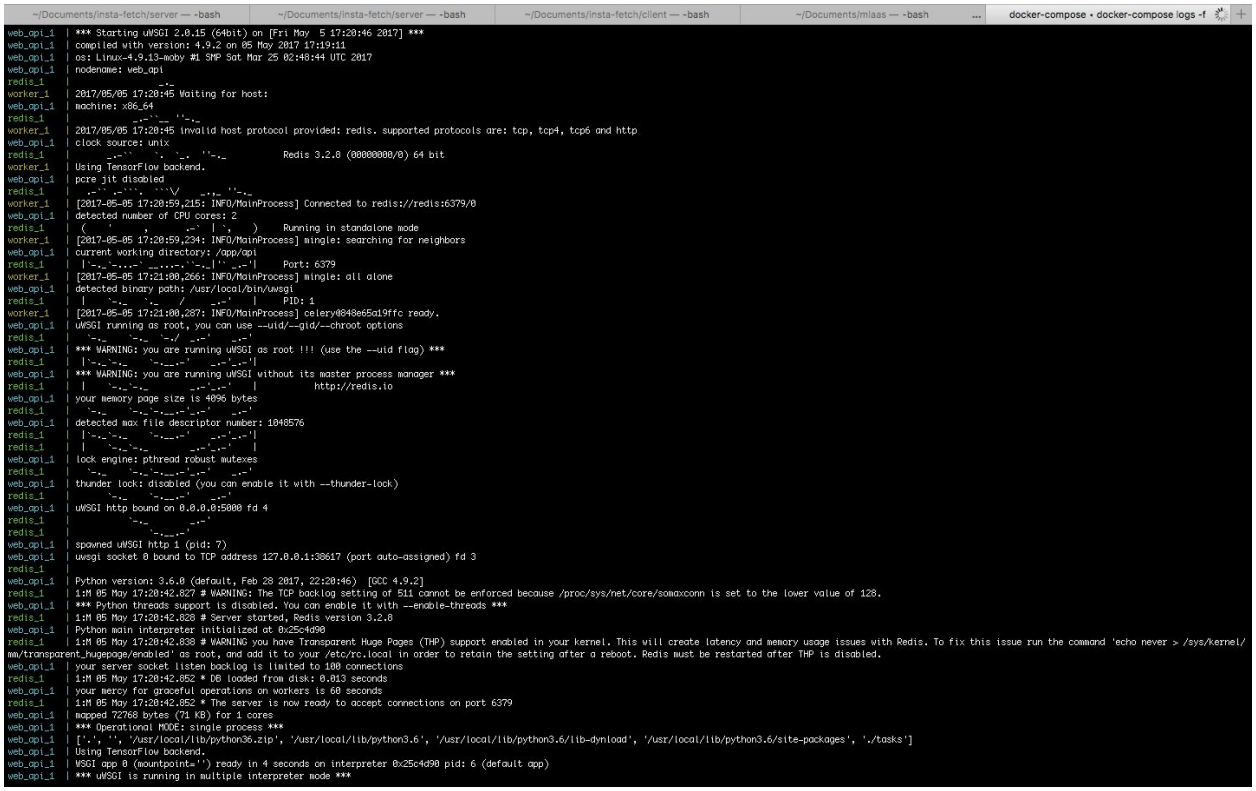
The distributed framework was designed to incorporate the use of Docker containers over traditional VM's. The use of containerization makes it easier to deploy and spawn copies of the instances so as to implement auto-scaling implementations. The current docker container application comprises of three interlinked images :-

1. **Redis**

Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries.

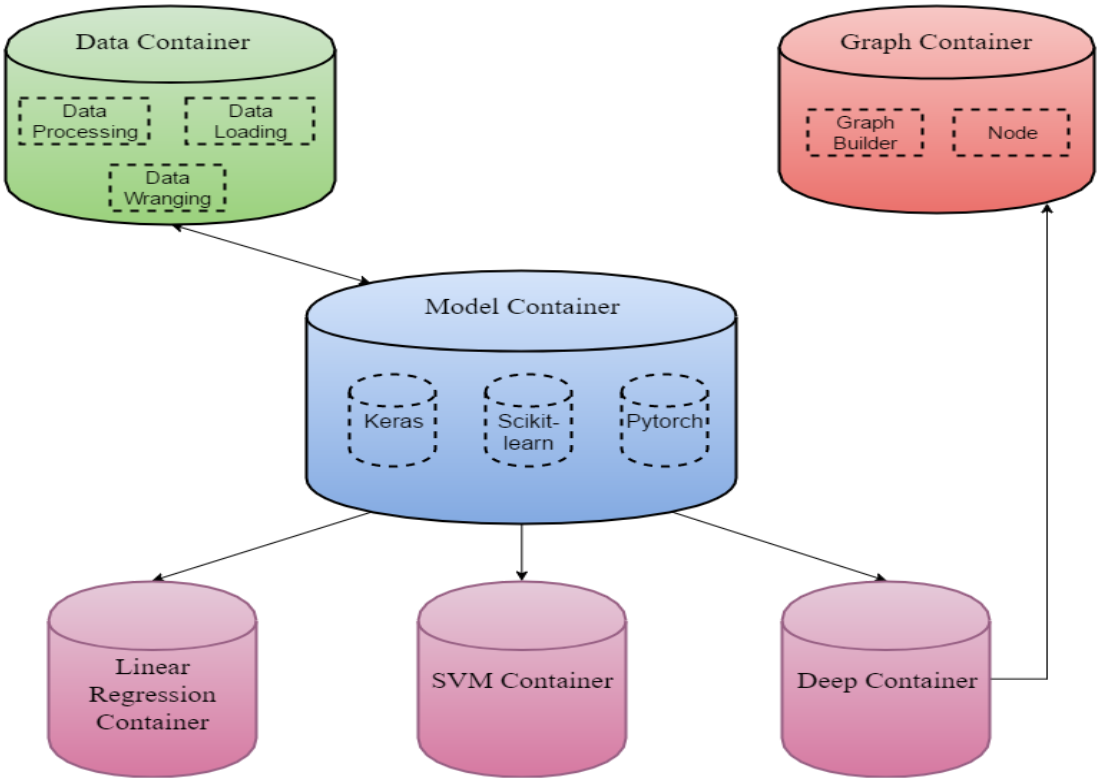
2. **WebAPI**
   The webAPI exposes a distributed cluster manager interface built on top of Celery, which implements an asynchronous model for distributed task/job queues based on distributed message passing.

3. **WorkerTask**
   The worker task image exposes a virtual container instance definition for the activities each worker node should perform. In our implementation each worker node can perform the following tasks:
   a. *train_model(objectid)*
   b. *predict_on_model(objectid)*



ii. Machine learning models are trained within the distributed framework. The framework has provisions to handle both standard datasets popularly used in the community as well as datasets provisioned by the user.
The manager establishes a connection to the shared database instance and pulls information about the model as specified by the user. Worker nodes are then used to create machine learning models from the information. The manager is designed to be agnostic to the data it receives from the shared database. This ensures that the all models are consistent and created the same way no matter how different the data is. The manager also routes the data to the worker nodes to be processed.

A schema of how the various entities interact with each other is shown below.



The Data Container is designed to take care of the data processing and loading pipeline including functionality for data wrangling in case the user's data is not in standard format. The Model Container has generic functionality for training and performing inference on data through more specific templates which depend on the use case of the user. The containers are library agnostic and use whichever one is best suited for the task at hand but currently we support Keras, and scikit-learn [8] with future support for PyTorch [9] coming soon. The Graph Container is used for building deep neural networks utilized by the Deep Container for training and prediction.

c. Model Query Framework:
   i. Once a model has been trained and tested, it is pushed to a naive model storage implementation in our shared database instance. The model storage component interacts deeply with the dynamic RESTful API server application. This component requires a dynamic nature due to the ever changing/updating behavior of machine learning model. Our framework aims to guarantee the availability of the latest trained model for querying through the RESTful API and hence our dynamic API application always

loads the latest trained model and exposes that to external requests at the discretion of the user on the client application.

d. Client/Iot Access:
   i. The external requests can be accessed by the user through a python script. Below is an example of a code snippet that trains a custom user-defined dataset. After the model is trained on our cloud framework, our API allows the predictions to be made via the Predict functions.

```
import requests
import json
from bs4 import BeautifulSoup
import time
import numpy as np
import sys

ip='http://54.214.195.161:5000/'
object_id="590b45c230adc0439dbacc21"
retval=False
data0={"objectid":object_id,"userdata":{'x':[[1],[2],[3]],"y":[[1],[2],[3]]}}
r = requests.post(ip+'train',json=data0)
r=r.text
r=r.split('href="')[1].split('">')[0].replace("\n","")
print(r)
while retval==False:
    time.sleep(2)
    temp=requests.get(r)
    ret_val=temp.text
    print(ret_val)
    if ret_val=="true":
        data1 = {"objectid":object_id,"predict_data":[[36],[54]]}
        r = requests.post(ip+'predict',json=data1)
        pred=r.text
        print(pred)
        pred=json.loads(pred)

        print("Predicted Value:")
        print(pred["predictions"])
        break
```

The MLaaS framework offers the ability to provision for on-demand compute instances which can be used to scale up and scale out resources as per user requirements to train models quickly without compromising on the accuracy.

4. Conclusion and Future Work:

We have described Mlaas, a machine learning framework on the cloud. The framework can load data process it and run machine learning models. This can be used by people from different professional backgrounds who have no machine learning or deep learning expertise to train their data thereby giving them the power of machine learning.

We also extend the scope of our application to account for some deep learning models. In case of deep learning models, we provide the user with the flexibility to use pretrained features like ImageNet or train their own model. These models require cloud instances with GPU support to tackle the computational requirement for the training tasks required for such models. We are interested in exploring how to extend our current implementation to tackle distributed training on multiple GPUs.

Additionally, we are planning to explore if the functionalities offered by our framework can be made as on-demand as the framework itself. Applications like Eucalyptus may assist us in scaling up or down the number of worker instances in order to cater to the user's time constraints.

6. Reference:

[1] E. Alpaydin, Introduction to machine learning. MIT press, 2014

[2] https://blog.keras.io/on-the-importance-of-democratizing-artificial-intelligence.html
https://predictionio.incubator.apache.org/start/

[3]A. Baldominos, E. Albacete, Y. Saez, and P. Isasi, "A scalable machine learning online service for big data real-time analysis," in Computational Intelligence in Big Data (CIBD), 2014 IEEE Symposium on. IEEE, 2014, pp. 1–8

[4] https://cloud.google.com/prediction/

[5] https://keras.io/

[6] https://www.tensorflow.org/about/bib

[7] http://deeplearning.net/software/theano/

[8] http://scikit-learn.org/stable/

[9] http://pytorch.org/