National College of Ireland

Efficiency of Machine Learning Cloud-Based Services vs Traditional Methods in Stock Prices Prediction

MSc Research Project
Cloud Computing

Siarhei Staravoitau
Student ID: x18162070

School of Computing
National College of Ireland

Supervisor: Horacio Gonzalez-Velez

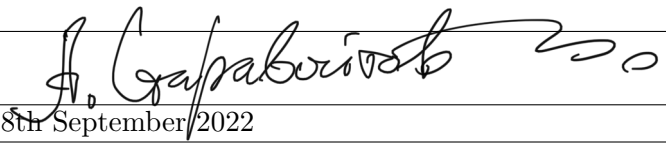
National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Siarhei Staravoitau
Student ID:	x18162070
Programme:	Cloud Computing
Year:	2021-22
Module:	MSc Research Project
Supervisor:	Horacio Gonzalez-Velez
Submission Due Date:	19/09/2022
Project Title:	Efficiency of Machine Learning Cloud-Based Services vs Traditional Methods in Stock Prices Prediction
Word Count:	6102
Page Count:	26

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	18th September 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Efficiency of Machine Learning Cloud-Based Services vs Traditional Methods in Stock Prices Prediction

Siarhei Staravoitau
x18162070

Abstract

One of the challenges while running ML jobs is IT compute resource usage optimisation. Scaling of computing resources and distributed calculations are in the mainstream of cloud computing development to optimize business IT resources and costs. This research evaluates the efficiency of cloud-based platforms, tools and features (PySpark scalability and distributed Tesnsorflow *MirroredStrategy*, *TPUStrategy* and Elephas RDD model training) over cloud-based platforms while performing Data Analytics task like LSTM Stock Price Prediction using sentiment analysis based on Twitter microblog messages. The RMSE (as well as MAE, MAPE, etc.) accuracy and model training time are used as benchmarks for the evaluation of cloud computing environment parameters. Changing model training hyperparameters (number of neurons, number of epochs, batch size) is used to change research tests' workload using Pandas and PySpark code implementation in Google Colab Pro+ GPU and TPU, AWS EMR, AWS Lambda and Local PC with GPU environments. Best RMSE accuracy and model training times were demonstrated by both PySpark-based and Pandas-based code executed using GPU in non-distributed model training mode. Distributed model training improves model training time, but model accuracy is getting reduced. Using the AWS EMR and AWS Lambda experiments along with the Google Colab-based Keras Tuner tool with Random search hyperparameters optimization didn't produce the expected RMSE accuracy and model training times.

1 Introduction

The development speed of cloud-based compute resources paced out the growth of machine learning requirements and creates a challenge how to select correct compute resources and optimize model training times and accuracy in fine-tuning model hyperparameters. The purpose of this research is to contribute to establishing to what extent Cloud-based Machine Learning Service is more efficient and if it could replace the on-prem approach while performing stock price prediction. Research Question:

> To what extent Machine Learning Cloud-Based Services can enhance the efficiency of stock price prediction and replace it to support investors and stock market traders?

The contribution of this project: establish and compare which cloud computing solution, including PySpark framework and distributed model training, is more efficient when

performing Data Analytics task like TensorFlow Sequential time series stock price prediction by measuring average execution time and accuracy (RMSE, MAE, MAPE, MDAPE) with different hyperparameters (batch size, epochs and number of neurons). The research is testing the efficiency of using Cloud Computing platforms (Google Colab Pro+ with GPU and TPU, Jupyter Lab), tools and features like PySpark scalability and TensorFlow distributed model training when performing LSTM Stock price prediction. The accuracy and model processing time of performing ML tasks are benchmarks for the evaluation of changing cloud computing environment parameters. Changing model training hyperparameters is used to change test workload for evaluation of defined Cloud Computing environments. And, of course, this research has a practical interest to measure performance with meaningful accuracy results and model training time that could be useful for real-life business use cases when defining tools and strategies for ML IT infrastructure and indicate which cloud-based tools could be used and which ones should be avoided from using in ML Cloud infrastructure. The project runs a set of experiments in Google Colab Pro + (both GPU and TPU enabled), AWS EMR, AWS Lambda and local PC having a grid test search strategy to establish which infrastructure configuration, model training approach and data set processing from selected for testing are more efficient when performing Machine Learning for Stock Prices prediction with sentiment analysis. The project's limitations are hardware and software compatibility, and college access compute resources infrastructure deploying. Research structure: review the state of the art of Machine Learning cloud-based solutions, platforms and frameworks which could be used for Stock price predictions, ML algorithms for using stock price prediction and sentiment analysis. The methodology part describes an approach for collecting experiment data, performing testing and results evaluation. The Evaluation section contains RMSE and model training tine test results from distributed and non-distributed model training solutions having fine-tuned models trained with different hyperparameters: batch size, number of epochs and number of neurons. Best RMSE accuracy and model training time is delivered by GPU-based solutions. RMSE of 1.99 and model training time of 167.73 s executed on PySpark solution in Google Colab Pro+ with 700 neurons, 100 epochs and 32 batch size. Distributed model training improves PySpark model training time and TPU optimized solution, but RMSE accuracy could be a trade-off for using distributed training. The conclusion part summarizes the research results and future work.

2 Related Work

2.1 Machine Learning Services and Platforms

The biggest challenge in the implementation of Machine Learning (ML) in a business process is to select the right ML framework and find a balance between on-prem and cloud implementation along with finding scalability and distributed calculations effective balance. Model training is the largest element of ML consuming IT resources, so it is necessary to find a trade off between compute elements and actual model training hyperparameters. Scaling up options could be generic GPU and Tensor Processing Unit (TPU) provided by Google and Azure FPGA [1, 2, 3]. ML model training efficiency could be evaluated by calculating RMSE as accuracy measure (as well as MAE, MAPE, MDAPE) [4] and model training time. User experience, ML infrastructure implementation and its complexity contribute to user ability deliver practical result. Depending on task and scale of business, it could be used on-prem, cloud-based and hybrid solutions, implemented on

a specialized ML services platforms or bespoke designed. Simplest to use with great user experience are Jupyter Lab based [5] solutions like Google Colab (GPU and TPU) [6]. AWS EMR [7] has an advantage of having Jupyter Lab Notebooks combined with Spark framework. Also, one of the options could be using of AWS Lambda[8] due to its instant availability and robustness. It is important to keep in mind data science legal and ethical norms[9], data privacy and protection of sensitive data[10, 11, 12], hacker attack risks to prevent model corruption[13], unauthorized access to the hosted model and data[9]. Python and Jupyter Notebooks are very popular among ML researchers due to their simplicity and convenience. The Table 1 summarizes platforms and services used in the research.

Table 1: Cloud-based Platforms and Services for ML

Google Colab Pro	Cloud-based Jupyter notebooks platform, allowing execute arbitrary python code in a web browser. Colab Pro + costs 50 Euro per month. Fast and expensive. GPU and TPU processing. But, Google could ban users from GPU access due to the fair use policy.[6]
AWS Lambda	On-demand cloud computing function-as-a-service. Runs code triggered by an event. Advantage: cheap to use, availability and robustness. Drawback: processing time 15 minutes, no GPU processing.[8]
AWS EMR	EMR is a cloud big data platform for running large-scale distributed data processing jobs, interactive SQL queries, and machine learning (ML). Advantages: work with big data, provisioning the required number of cluster nodes. Drawbacks - cost factor, PySpark kernel is not compatible with Pandas.[7]

To improve model training time it distributed model and data training could be used, where workload is split and shared among multiple worker nodes. Distributed data training assumes splitting data set in chunks and send it to worker nodes for training(Figure 1). Distributed model training segments model into different parts and runs them concurrently on different workers, which need to synchronize on shared parameter server(Figure 2)[14, 15]. Tensorflow offers following strategies for distributed model training: *MirroredStrategy, TPUStrategy, MultiWorkerMirroredStrategy, CentralStorageStrategy, ParameterServerStrategy, Default Strategy and OneDeviceStrategy* [16]. *MirroredStrategy* and *TPUStrategy* are most relevant for this research as they could be configured and tested both on Google Colab and Local PC including model training on a single PC with CPU and GPU devices in a same time. *TPUStrategy* allows to test Google's TPU distributed model training on Google Colab . *MultiWorkerMirroredStrategy and ParameterServerStrategy* could be used for multiworker distributed model training [16]. Spark Elephas Resilient Distributed Data[17] model training also could be used for multiworker distributed model training. It allows to spin up a number of ML workers and train the Keras model in asynchronous mode.[17] and configured to run training jobs both on a cluster, server-less solution or set of virtual hosts on a local High performance PC. The trade off when using distributed training could be model accuracy. Tensorflow, Apache Hadoop, Apache Spark frameworks functionality allows to perform

distributed both data and model calculations using various topology [3, 2, 1]. Scaling compute operations with big a data could be done on AWS EMR virtual clusters allowing to process of large data sets using PySpark SQL.

Among distributed *data-parallel* model training could be highlighted Horovod [15] on Figure 1. Also, Figure 2 demonstrates ML Distributed parameter server options model training approach. TensorFlow distributed model training is using similar approach [1].

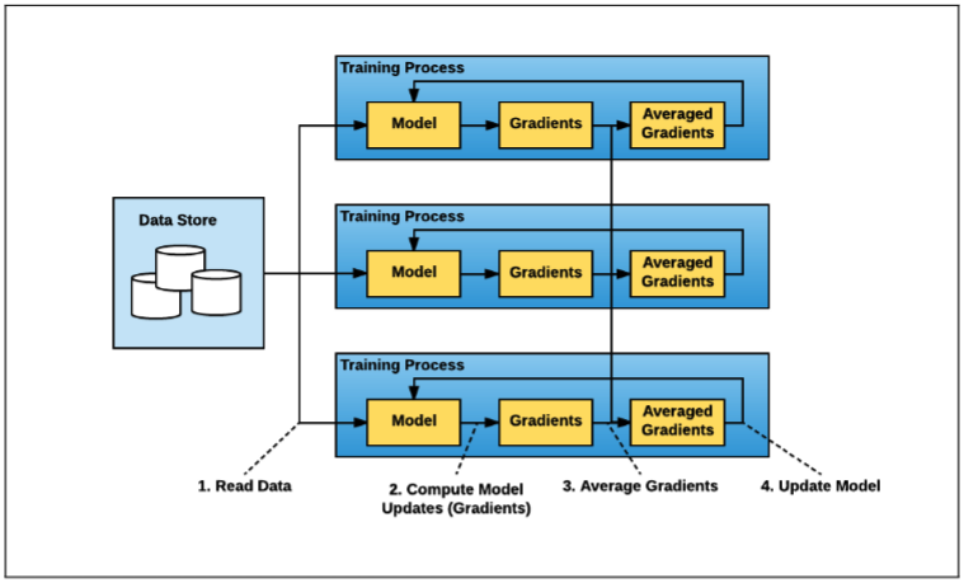


Figure 1: ML Distributed parallel data training approach

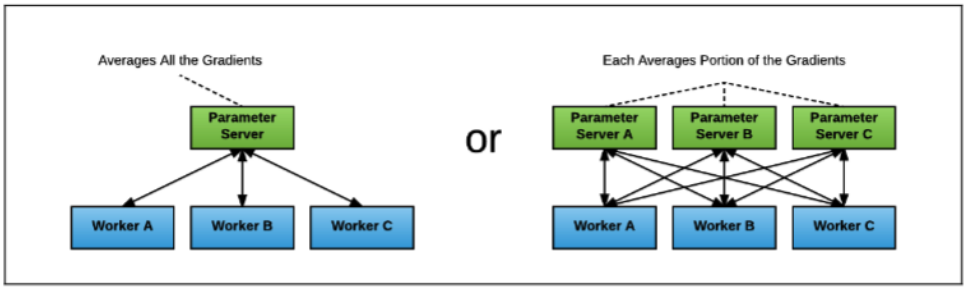


Figure 2: ML Distributed parameter server options

2.2 Machine Learning Techniques for Stock Prices Prediction

Due to stock market volatility caused by world politics and other influencing factors, it is a very challenging task to perform accurate stock price predictions.[18] The Deep Learning ML approach and Recurrent Neural Networks are very popular for predicting time series data using long-and-short-term memory LSTM networks[19]. LSTM networks are correcting long-term series data by applying stochastic gradient descent using Adam, Adamax, Nadam, AMSGrad, etc. optimizations. LSTM model consists of the following components: (a) an input gate, (b) a self-recurrent connection neuron, (c) *forget* gate proposed by Ger et al. [20], and (d) an output gate. This model structure allows addressing an issue of vanishing gradient. Nelson et al. [21] in their research confirmed

that forecasting (LSTM) delivers better results in comparison with other techniques as the LSTM model gives fewer risks. For time series forecasting could be used TensorFlow Sequential model [22]. It provides user-friendly instruments to configure model layers. To optimize model training time and accuracy could be used fine-tune hyperparameters like the number of epochs, batch size, and number of neurons.[23] Abbasimehr et al. emphasize that selection of a correct number of neurons helps to prevent model overfitting [23]. As accuracy measures could be used RMSE (as well as MAE, MAPE, and MPADE) [24]. Also, one of the efficiency measures is model training time. A number of neurons are defined by multiplying the number of features and lag. Also, for model optimizing could be used Keras Tuner simplifies the hyperparameter search.[25]

2.3 Sentiment Analysis Techniques for Stock Prices Prediction

The research project uses a sentiment analysis of Twitter micro-blog messages related to selected stock security. It calculates polarity index for each day using the Vader framework[26] and added as an engineered feature into the daily stock prices data set for further processing[27]. Table 2 combines a research idea to use Sequential LSTM model, sentiment analysis, hypeparameters fine-tuning and model training on distributed and scalable solutions.

Table 2: Proposal for conducting Research

Approach	ML Framework	Sentiment Analysis	Distribute model Training	LSTM Hyperparameters fine-tuning on distributed model and data training
LSTM [19, 21, 20, 22, 27]	✓			
LSTM Hyperparameters tuning [23, 26]	✓			
Vader Framework Sentiment Analysis [26]	✓	✓		
Distributed model training [17, 28, 1, 29]	✓	✓	✓	
Proposal	✓	✓	✓	✓

2.4 Conclusion

The main idea of this paper is to establish the most efficient cloud-based solution delivering the best accuracy and model training time while performing Stock Price prediction task implemented on selected cloud-based platforms. This involves the design, implementation, code execution in a cloud scalable environment, having ML model training and processing data in *MirroredStrategy*, *TPUStrategy* distributed and Elephas resilient distributed mode and comparing the model training time and RMSE accuracy of a Sequential Time Series model for stock price prediction having as one of the features a

daily sentiments polarity indexes based on Twitter microblog messages. Considered implementation solutions are based on Google Colab Pro + Jupyter notebooks (GPU and TPU), AWS EMR, AWS Lambda frameworks and local High-Performance PC solutions using GPU. Data processing considerations are using Pandas and Spark SQL. For stock price prediction are used LSTM time series TensorFlow Sequential Model is adopted in all test implementations, including Keras Tuner and Elephas framework. The efficiency of each experiment is evaluated based on average values of model accuracy RMSE (as well as MAE, MAPE, MDAPE) and model training time. The novelty of this paper is finding the most efficient cloud-based solution delivering the best accuracy and model training time and possible trade-off based on a comparison of execution results from distributed and non-distributed GPU, CPU and TPU-based solutions running on Spark-based and Pandas-based solutions including fine-tuning of hyperparameters. Questions to be answered:

1. Which platform is delivering the lowest RMSE and model training time when using Pandas-based code or PySpark-based code in *MirroredStrategy* and *TPUStrategy* distributed and non-distributed model training mode?

2. How accurate and fast and Elephas RDD distributed model training?

3. How accurate and fast is Keras Tuner model training?

3 Methodology

3.1 Data set preparation and features engineering

The project uses the data mining technique to do in-depth enquiries into stock prices and Twitter[30] data to deliver a meaningful set of data for predicting future trends. Workflow for ML Stock prices prediction contains the following stages:

1. Data collection

2. Cleaning, calculate daily polarity indexes, preparing a data set for training

3. Export combined data set into CSV file.

ETL Jupyter Notebook creates aggregated data set combining stock price data from Yahoo Finance[31] with calculated daily average polarity index of tweet messages for the period between 24-July-2017 - 21-July-2022 GOOG ticker and #GOOG hash tag. Twitter micro-blog English language tweets are retrieved using Twarc2 and Twitter academic developer account credentials. Daily polarity indexes are calculated using Vader framework. Both stock prices and daily average polarity indexes are aggregated in Pandas data frame and exported into a CSV file to AWS S3 bucket *s3://mscloudetop-etl/*. All data retrieval and data set preparation done on a Google Colab using Python3, Pandas, Numpy, and Vader libraries. Figure 4 outlines schema of loaded combined data set into both Pandas and PySpark data frame.

3.2 Tests execution and processing results

Tests are executed three times and from received figures calculated average values and standard deviation values and documented into report tables. For evaluation of test execution efficiency *model training time* is used. As accuracy benchmark is used RMSE (where n - number of observations; y - predicted values; x - actual values):

$$RMSE = \sqrt{(\frac{1}{n})\sum_{i=1}^{n}(y_i - x_i)^2}$$

When review model training time results, only selected results with lowest RMSE values in order to eliminate bad test results. For test cases processed in AWS EMR, AWS Lambda and Google Colab TPU a fixed number of neurons to 350 was set in order to mitigate costs and function crashes.

4 Design Specification

Main requirement of this research is to compare ML model training time and accuracy in implementation variants on Figure 3. Project code is split in two main parts: ETL part and Model training and prediction. ETL Part performs data preparation for the Model training and prediction.

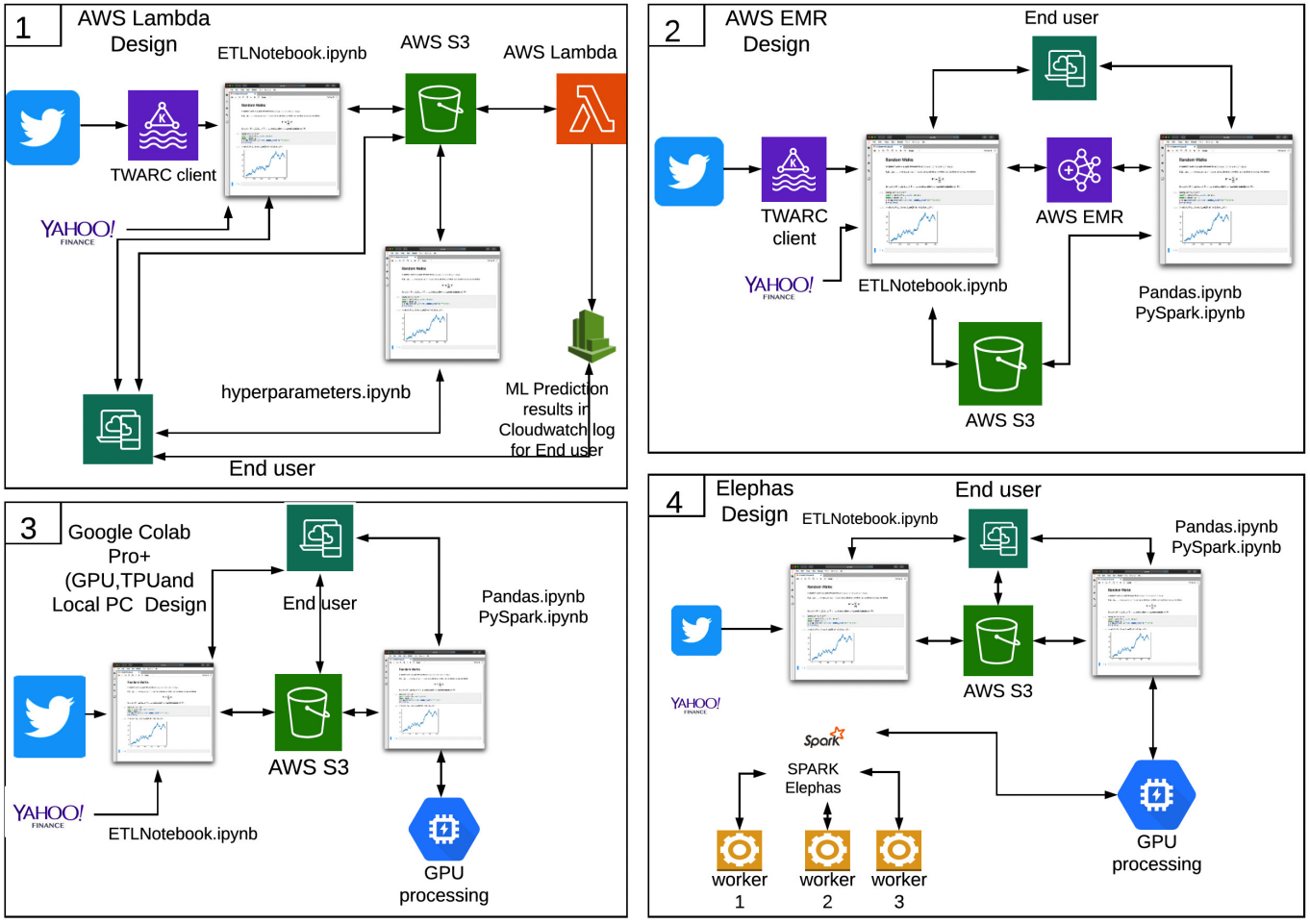


Figure 3: ML Execution Flows

1. AWS Lambda design on Figure 3 slide 1 outlines execution flow based on ETL Jupyter notebook, hyperparameters and results Jupyter notebook, AWS S3, AWS Lambda and AWS CloudWatch setup performing data collection, features engineering by *ETLNotebook.ipynb*, LSTM model training and forecasting using AWS Lambda. AWS Lambda function is invoked by AWS S3 bucket trigger when new CSV file is saved into AWS S3 bucket *mscloudetop-etl*. Hyperparameters configuration is managed via *hyperparameters.ipynb* saving file *hyperparameters.csv* with updated hyperparameters to the AWS S3 bucket *mscloudetop-etl*. User can update number of epochs, batch size, sequence length. AWS Lambda Execution results could be accessed either by running a script in Jupyter Notebook *ETLNotebook.ipynb* to read *results.csv* file from *msccloudetop-results* bucket, accessing Cloudwatch logs or downloading file directly from the this bucket.

2. AWS EMR design on Figure 3 slide 2 outlines execution flow for tests based on AWS EMR Jupyter notebook. ETL Jupyter notebook *ETLNotebook.ipynb* saves combined data set *stock price.csv* in *mscloudetop-etl*. There are two separate Jupyter Notebooks of ML scripts using PySpark and Pandas data frame. EMR cluster is based on *c5.24xlarge* instance. Prediction results are saved in *msccloudetop-results* bucket.

3. Google Colab Pro + on Figure 3 slide 3 outlines an execution flow based on Google Colab Jupyter Pro+ notebook, with GPU/TPU and extended memory setup performing data collection by running ETL Jupyter notebook *ETLNotebook.ipynb*. Combined data set file is saved into AWS S3 bucket *mscloudetop-etl*. ML model training hyperparameters and actual model training script is executed by *ML Model training and prediction Jupyter Notebook*. Results are delivered to the end user in a same notebook and saved into CSV file in the *msccloudetop-results* bucket. This platform is used for both Pandas-based and PySpark-based Jupyter Notebooks. Tests here are executed using both GPU and TPU options, distributed and non-distributed mode. Keras Tuner tests are executed using Google Colab Pro + GPU as well.

4. Spark Elephas on Figure 3 slide 4 outlines tests based on Google Colab Pro+ Jupyter notebook, with GPU and extended memory. ETL Notebook *ETLNotebook.ipynb* performs data set preparation. Number of workers are configured at PySpark application level. Elephas framework performs distributed model training and delivers results to the end user into CSV file in *msccloudetop-results* AWS S3 bucket and in Jupyter notebook.

5 Implementation

Project code design and implementation are done using Jupyter notebooks. There are two main parts:

- ETL Jupyter Notebook. ETL Jupyter Notebook *ETLNotebook.ipynb* is based on Python 3. Jupyter Notebook logic performs tweets messages retrieval for specified security symbol, tweets hashtag and analysis period using Twarc2 client from Twitter, cleans text data set using Regex and prepares data set for polarity indexes calculation using nltk framework Vader lexicon. For research purposes, it was tweets

are acquired using Twitter academic account. Stock prices are retrieved by using Yahoo Finance API.

The test period is set to 5 years from 24/07/2017 until the date 21/07/2022. It could be taken a larger test period from the initial company IPO, but due to the significant price increase, it could bring data anomalies causing a negative effect on training and prediction values. Yahoo Finance historical data are normally accurate and does not require additional cleaning and validation, but at the same time, the preparation of tweet messages requires cleaning to have only words in the data set. Calculated polarity indexes are merged with stock prices data are saved to AWS S3 for the next stage of the ML process. For model training was selected following features: *High*, *Low*, *Open*, *Close*, *Volume*, *compound*, *Adj Close*. The target for model training is *Close* (Figure 4). Stock prices data and polarity indexes are saved to AWS S3 bucket *s3://mscloudetop-etl/tweets/* , *s3://mscloudetop-etl/tweets/merged/* and final dataset file *stock_price.csv* saved to *s3://mscloudetop-etl/*.

Figure 4 outlines schema of loaded combined data set into data frame.

```
root
 |-- _c0: integer (nullable = true)
 |-- Date: timestamp (nullable = true)
 |-- Open: double (nullable = true)
 |-- High: double (nullable = true)
 |-- Low: double (nullable = true)
 |-- Close: double (nullable = true)
 |-- Adj Close: double (nullable = true)
 |-- Volume: integer (nullable = true)
 |-- compound: double (nullable = true)
```

Figure 4: Data set schema.

- ML Job Jupyter Notebooks perform data import file from the previous stage from *stock_price.csv* in *s3://mscloudetop-etl/* AWS S3 bucket. Data visualization is done using matplotlib and seaborn. Performance measuring and data scaling are done using sklearn. Data handling and processing are done by using either Pandas Data frame or Spark Data frame and data set split 80% for training and 20% validating the model. For stock price forecasting used the TensorFlow Keras Sequential LSTM model with optimizer Adam and loss mse. The trained model and predicted price, accuracy and error results are saved into the AWS S3 bucket. Some implementation code ideas were used from [32]. Following variants of Jupyter Notebooks are created:

  1. Jupyter Notebook *Pandas-based-Colab.ipynb* model training and prediction
  2. Jupyter Notebook *PySpark-based-Colab.ipynb* model training and prediction
  3. Jupyter Notebook *EMR-Pandas-based-Colab.ipynb* model training and prediction
  4. Jupyter Notebook *EMR-PySpark-based-Colab.ipynb* model training and prediction
  5. Jupyter Notebook *Elephas.ipynb* distributed model training.
  6. Jupyter Notebook *Keras-Tuner-Colab.ipynb* model training and prediction

Model training hyperparameter settings:

1. Model training on Google Colab Pro+ and local High Power PC for all hyperparameter combinations including number of neurons (210, 350, 700), batch size (8, 16, 32, 64, 128) and number of epochs (30, 50, 100)

2. Model training with fixed number of neurons (350), and changing number of epoch (30, 50, 100) and batch size (8, 16, 32, 64, 128) on AWS EMR and AWS Lambda implementations due to cost factor and limitations.

3. Keras Tuner model training on Google Colab and local High Power PC with fixed number of neurons (350) and pre-defined range of batch size in Keras model RandomSearch parameters (from 8 to 512).

4. Elephas RDD model training on Google Colab Pro with fixed number of neurons (350), changing number of workers (1, 2, 3), batch size (8, 16, 32, 64, 128) and number of epochs (30, 50, 100)

It is used following model code for all tests illustrated on Figure 5. Dense parameter is left unchanged.

```
model = Sequential()
model.add(LSTM(n_neurons, return_sequences=True, input_shape=(x_train.shape[1], x_train.shape[2])))
model.add(LSTM(n_neurons, return_sequences=False))
model.add(Dense(5))
model.add(Dense(1))
model.compile(optimizer='Adam', loss='mse', metrics=['accuracy','binary_accuracy','MSE','MAPE','MAE'])
```

Figure 5: LSTM model code

Train and test data are split with ratio 80:20 and scaled with MinMaxScaler from sklearn. Model optimizer is Adam. Model with layers 700 neurons is illustrated on Figure 6

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 100, 700)          1982400

 lstm_1 (LSTM)               (None, 700)               3922800

 dense (Dense)               (None, 5)                 3505

 dense_1 (Dense)             (None, 1)                 6

=================================================================
Total params: 5,908,711
Trainable params: 5,908,711
Non-trainable params: 0
```

Figure 6: Model Map with 700 neurons

All tests are executed three times and average figures and standard deviation are calculated for model training time, script execution time, RMSE, MAE, MAPE, MDAPE, change per cent. Implementation environments:

- Google Colab Pro + with GPU and extended memory
- AWS EMR one instance based on *c5.24xlarge*.
- AWS Lambda
- Mac Book Pro M1 MAX 14", 32 GB RAM, 24 GPU cores, 1 GB HD

6 Evaluation

6.1 Experiment 1. Accuracy Testing Results

6.1.1 Google Colab GPU and Local PC Test Results

Top 6 results executed using GPU with lowest RMSE values are aggregated in Table 3. The best average RMSE value of 1.99 (standard deviation 0.01) is received on PySpark-based Colab implementation with 100 epochs, 700 neurons, and batch size 32. Test with 700 neurons and batch size combination 100 and 32 vs 50 and 16 are producing the lowest RMSE values. Increasing the number of neurons, and epochs and decreasing batch size is increasing model training time. Among the top performers, there are two tests using *TesnsorFlow MirroredStrategy* distributed model training both executed on Colab and Local PC with RMSE of 2.05. Five out of six tests with the best results are obtained using PySpark. Model average training time is 167.7 seconds (STDEV 8.432). Very close accuracy results were demonstrated by Pandas-based Local PC (700 neurons, 100 epochs, batch size 32) with RMSE 2.03. Also, RMSE of 2.01 is observed on a PySpark Local PC (700 neurons, 50 epochs, batch size 16) (Table 3). Table 4 aggregates Top 6 test run results executed using using distributed *MirroredStartegy*. Distributed model training lowest RMSE of 2.05 (Stdev 0.57) with model training time shows PySpark-based code Colab Pro+ GPU (350 neurons, 50 epochs, 16 batch). The same accuracy demonstrates Pandas-based code executed on Local PC (350 neurons, 100 epochs, 32 batch) with 156.11 s model training time. Tests show that both Pandas-based and PySpark code deliver very close RMSE values, but using distributed model training produces less accurate RMSE.

Table 3: Top 6 Test Run Results (using GPU) RMSE measures for GOOG Alphabet Inc. Price Prediction as of 22.07.2022

Experiment Name	PySpark Colab	PySpark LocalPC	PySpark LocalPC	PySpark Colab	Pandas LocalPC	PySpark Colab
DataFrame	PySpark	PySpark	PySpark	PySpark	Pandas	PySpark
Strategy	No	No	No	Yes	Yes	No
Number of Epochs	100	50	100	50	100	100
Number of Neurons	700	700	700	350	350	350
Batch Size	32	16	32	16	32	32
Predicted Price	115.14	115.63	114.65	114.96	114.56	114.89
Price Today	115.04	115.04	115.04	115.04	115.04	115.04
Change Percent	0.09	0.51	-0.35	-0.07	-0.42	-0.13
MAE	1.43	1.49	1.42	1.47	1.42	1.48
MAPE	1.1	1.15	1.09	1.12	1.08	1.13
MDAPE	0.82	0.86	0.77	0.8	0.74	0.81
RMSE	1.99	2.01	2.03	2.05	2.05	2.06
STDEV RMSE	0.01	0.02	0.1	0.57	0.05	0.09
Model Training time	167.73	212.69	244.53	53.77	156.11	64.83

Table 4: Top 6 RMSE Test Run Results (using distributed MirroredStartegy) for GOOG Alphabet Inc. Price Prediction as of 22.07.2022

Experiment Name	PySpark Colab	Pandas Local PC	PySpark Local PC	PySpark Local PC	Pandas Local PC	Pandas Local PC
DataFrame	PySpark	Pandas	PySpark	PySpark	Pandas	Pandas
Number of Epochs	50	100	50	30	50	100
Number of Neurons	350	350	700	350	700	700
Batch Size	16	32	16	8	16	32
Predicted Price	114.96	114.56	115.35	115.34	114.71	114.65
Price Today	115.04	115.04	115.04	115.04	115.04	115.04
Change Percent	-0.07	-0.42	0.27	0.26	-0.29	-0.34
MAE	1.47	1.42	1.54	1.57	1.53	1.52
MAPE	1.12	1.08	1.18	1.2	1.16	1.16
MDAPE	0.8	0.74	0.93	0.92	0.81	0.81
RMSE	2.05	2.05	2.07	2.09	2.12	2.13
STDEV RMSE	0.57	0.05	0.05	0.06	0.23	0.28
Model Training time	53.77	156.11	223.94	134.2	244.1	278.74

6.1.2 Google Colab TPU Testing Results

Tables 5 and 6 aggregate top 6 test executions with fixed number neurons 350 having lowest RMSE results for Pandas-based and Pyspark-based code implementation processed both with distributed TensorFlow *TPUStrategy* enabled and without distributed training. Best RMSE of 2.35 (Stdev 0.5) and model training time of 225 seconds result for distributed model training shows Pandas-based code (100 epochs, batch 32). For non-distributed model training best RMSE of 2.16 (Stdev 0.07) and model training time of 962.38 shows Pandas-based code (100 epochs, batch 32) as well. Distributed *TPUStrategy* model training shows better model training times over the non-distributed run, but RMSE accuracy in distributed training run declines.

Table 5: Top 6 Test Run Results in Google Colab TPU Distributed Distributed *TPUStrategy* Model training for GOOG Alphabet Inc. Price Prediction as of 22.07.2022, 350 Neurons

Experiment	Pandas Colab	Pandas Colab	Pandas Colab	PySpark Colab	PySpark Colab	Pandas Colab
Epochs	100	100	30	50	50	100
Batch size	32	8	16	8	16	64
Change Percent	0.65	0.12	0.69	-1.15	0.81	-0.19
MAE	1.86	1.81	1.99	1.98	2.17	2.04
MAPE	1.42	1.36	1.53	1.5	1.64	1.56
MDAPE	1.21	1.09	1.26	1.18	1.44	1.22
RMSE	2.35	2.4	2.54	2.57	2.67	2.67
STDEV RMSE	0.5	0.61	0.14	0.47	0.51	0.13
Model Training time, s	225.03	524.74	94.98	262.46	162.6	149.62

Table 6: Top 6 Test Run Results in Google Colab TPU Not Distributed Model training for GOOG Alphabet Inc. Price Prediction as of 22.07.2022, 350 Neurons

Experiment	Pandas Colab	PySpark Colab	Pandas Colab	Pandas Colab	PySpark Colab	PySpark Colab
Epochs	100	100	50	30	100	100
Batch Size	32	32	16	8	8	16
Change Per cent	0.16	-0.52	-0.79	0.37	-0.11	1.1
MAE	1.63	1.57	1.78	1.9	1.9	1.96
MAPE	1.24	1.19	1.35	1.44	1.44	1.49
MDAPE	0.98	0.84	1.05	1.21	1.17	1.31
RSME	2.16	2.19	2.37	2.41	2.46	2.46
STDEV RSME	0.07	0.25	0.39	0.23	0.81	0.6
Model Training Time, s	962.38	948	632.59	808.13	1778.9	1333.67

6.1.3 AWS EMR Testing Results

From tests executed on AWS EMR cluster with 350 neurons, the lowest average RMSE of 2.18 was received on Pandas-based code batch size 8 and 100 epochs. PySpark-based code showed an average RMSE of 2.19 on batch size 16 and the number of epochs 100. Model training time for batch size 8 is 834.19 seconds and for batch size 16 is 504.48 seconds.(Table 7). Testing on AWS EMR is time consuming and expensive.

Table 7: GOOG Alphabet Inc. Price Prediction Test Run Results as of 21.07.2022(AWS EMR, 350 Neurons, Batch Size 8 and 16; 100 Epochs)

Experiment Name	Pandas EMR	PySpark EMR
Batch Size	8	16
Number of Neurons	350	350
Number of Epochs	100	100
MAE	1.63	1.63
RMSE	2.18	2.19
RMSE STDEV	0.17	0.18
Model training time, s	834.18	504.48

6.1.4 AWS Lambda Testing Results

AWS Lambda test execution results are aggregated in Table 8. All tests were executed with a fixed number of neurons 350. The lowest RMSE value of 2.5 is produced by test runs with batch sizes 8, and 30 epochs. Test execution time is 818.7 seconds, which is very close to the AWS Lambda execution time limit. Test runs with 50 epochs, batch size 8; 100 epochs and batch size 8, 16 and 32 failed to complete due to execution time limits. AWS Lambda is not suitable for intensive ML processing jobs. It could have some ML potential if it would be based on GPU processing.

Table 8: GOOG Alphabet Inc price prediction test run results as of 22.07.2022 (AWS Lambda, 350 Neurons)

Number of Epochs	30	30	30	30	30	50	50	50	50	100	100
Batch Size	8	16	32	64	128	16	32	64	128	64	128
MAE	1.88	2.04	2.52	2.91	3.43	2.37	2.53	2.46	3.2	2.12	2.73
RMSE	2.5	2.64	3.27	3.73	4.34	2.89	3.14	3.21	4.06	2.77	3.49
RMSE STDEV	0.44	0.57	0.21	0.14	0.36	0.86	0.39	0.11	0.36	0.13	0.18
Time, s	818.7	443.97	315.62	242.4	201.81	736.59	526.25	393.19	317.15	798.63	631.67

6.2 Experiment 2. Model Training Time Testing Results

All test results are demonstrating a consistent trend to increase model training times with an increased number of epochs and decrease model training time with increasing batch size. Among tests demonstrated the lowest RMSE values (700 neurons, batch size 32 and number of epochs 30) executed 167.73 seconds in Google Colab vs 244.53 seconds in MacBook Pro. Decreasing the number of batches from 32 to 16 on MacBook Pro demonstrates an increase in processing time from 244.53 to 401.27 seconds or 39%. When comparing Pandas-based vs PySpark-based code in the Google Colab Pro+ environment, it appears that Pandas-based code is training model faster. Example of test executions with fixed number of neurons of 350 is on Figure 7. But, in the AWS EMR environment

with a fixed number of neurons of 350, test run results on Figure 8 demonstrate that both Pandas-based and PySpark-based code execution times are very close per batch size and per epochs run. That could be explained that both tests are executed on non-GPU-based instances.

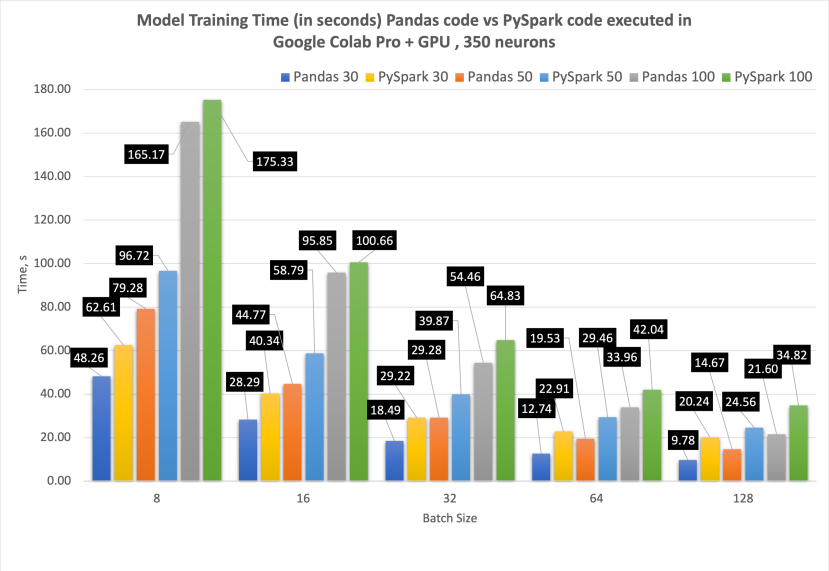


Figure 7: Model Training Time (in seconds) Pandas code vs PySpark code executed in Google Colab Pro + GPU , 350 neurons

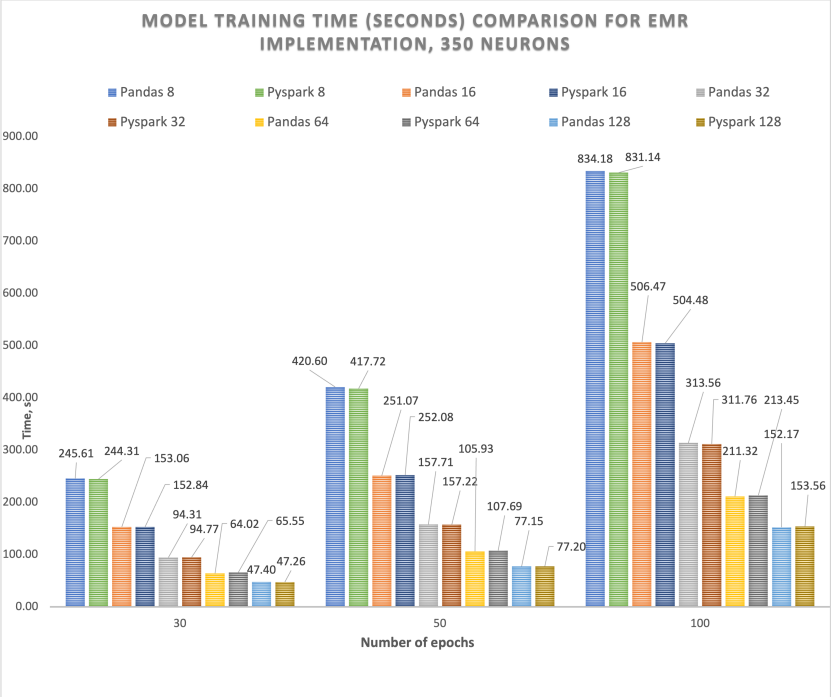


Figure 8: Model training time (seconds) comparison for EMR implementation, 350 neurons

Table 3 aggregates model training time for the top 6 test runs with the lowest RMSE values. The fastest model training time of 64.83 s demonstrates PySpark-based code ex-

ecuted on Google Colab using GPU (350 neurons, 100 epochs, batch size 32). Having the same hyperparameter settings, PySpark Local PC model training time is 120.41 s, i.e., Google Colab is 46.16% faster. Tables 9 and 10 demonstrate very interesting results for tests executed with TensorFlow *MirroredStrategy* in Google Colab Pro+ GPU and compared with non-distributed model training times. It takes a longer model training time for Pandas-based code when using distribute training mode, but TensorFlow *MirroredStrategy* reduces model training time for PySpark-based code, which could be explained by scaling of Spark Application.

Table 9: Distributed vs Non-distributed Pandas-based Code Model Training Time in Google Colab Pro+ GPU

Neurons	Epochs	Batch	No Strategy Time, s	Mirrored Strategy Time, s	difference	%
350	30	8	48.26	73.36	-25.1	-52.01
350	30	16	28.29	46.54	-18.25	-64.51
350	30	32	18.49	32.94	-14.45	-78.15
350	30	64	12.74	23.82	-11.08	-86.97
350	30	128	9.78	19.49	-9.71	-99.28
350	50	8	79.28	119.18	-39.9	-50.33
350	50	16	44.77	70.45	-25.68	-57.36
350	50	32	29.28	44.02	-14.74	-50.34
350	50	64	19.53	30.02	-10.49	-53.71
350	50	128	14.68	25.7	-11.02	-75.07
350	50	128	118.78	214.28	-95.5	-80.4
350	100	8	114.88	124.52	-9.64	-8.39
350	100	16	68.79	82.02	-13.23	-19.23
350	100	32	40.31	53.82	-13.51	-33.52
350	100	64	26.29	44.01	-17.72	-67.4
700	30	8	138.44	160.99	-22.55	-16.29
700	30	16	76.78	90.75	-13.97	-18.19
700	30	32	38.69	57.71	-19.02	-49.16
700	30	64	33.12	41.46	-8.34	-25.18
700	30	128	25.63	33.94	-8.31	-32.42
700	50	8	228.63	262.54	-33.91	-14.83
700	50	16	126.74	147.9	-21.16	-16.7
700	50	32	67.3	91.16	-23.86	-35.45
700	50	64	53.52	65.12	-11.6	-21.67
700	50	128	40.7	51.55	-10.85	-26.66
700	100	8	454.46	518.36	-63.9	-14.06
700	100	16	157.29	287.27	-129.98	-82.64
700	100	32	96.31	176	-79.69	-82.74
700	100	64	66.94	123.06	-56.12	-83.84
700	100	128	52.21	95.36	-43.15	-82.65

Table 10: Distributed vs Non-distributed PySpark-based Code Model Training Time in Google Colab Pro+ GPU

Neurons	Epochs	Batch	No Strategy Time, s	Mirrored Strategy Time, s	difference	%
350	30	8	62.61	61.21	1.4	2.24
350	30	16	40.34	35.33	5.01	12.42
350	30	32	29.22	25.28	3.94	13.48
350	30	64	22.91	18.66	4.25	18.55
350	30	128	20.24	15.55	4.69	23.17
350	50	8	96.72	92.55	4.17	4.31
350	50	16	58.79	53.77	5.02	8.54
350	50	32	39.87	37.64	2.23	5.59
350	50	64	29.46	26.13	3.33	11.3
350	50	128	24.56	20.78	3.78	15.39
350	100	8	175.33	183	-7.67	-4.37
350	100	16	100.66	99.29	1.37	1.36
350	100	32	64.83	64.45	0.38	0.59
350	100	64	42.04	42.22	-0.18	-0.43
350	100	128	34.81	33.49	1.32	3.79
700	30	8	154.48	92.87	61.61	39.88
700	30	16	89.56	60.46	29.1	32.49
700	30	32	57.68	39.97	17.71	30.7
700	30	64	43.71	29.72	13.99	32.01
700	30	128	36.31	24.72	11.59	31.92
700	50	8	251.68	151.05	100.63	39.98
700	50	16	141.59	94.38	47.21	33.34
700	50	32	88.81	61.56	27.25	30.68
700	50	64	64.43	45	19.43	30.16
700	50	128	51.13	37.54	13.59	26.58
700	100	8	488.75	293.71	195.04	39.91
700	100	16	268.93	182.61	86.32	32.1
700	100	32	167.73	116.33	51.4	30.64
700	100	64	124.97	83	41.97	33.58
700	100	128	87.38	65.46	21.92	25.09

6.3 Experiment 3. Elephas framework Multi Worker Testing

Elephas framework GOOG Alphabet Inc testing results with a fixed number of neurons 350, batch size 8, 16, 32, 64, 128 and number of workers 1, 2 and 3. All tests are executed using Google Colab Pro + with GPU and extended memory. One worker test executions with top 5 RMSE values are aggregated in Table 14. The most precise results are shown for one worker (350 neurons, batch size 8, 50 epochs) with RMSE 2.05 (STDEV 0.09) and model training time 55.25 seconds. The predicted price change per cent is 0.4%, which is very close to the actual price.

Table 11: Elephas framework GOOG Alphabet Inc price prediction test run results as of 21.07.2022(350 neurons, one worker)

Number Of Workers	1	1	1	1	1
Batch Size	8	16	16	32	64
Number of Epochs	50	30	100	100	100
Predicted Price Change %	0.4	0.07	-0.09	0.25	0.31
MAE	1.48	1.75	1.67	1.91	2.08
RMSE	2.05	2.34	2.22	2.48	2.61
RMSE STDEV	0.09	0.12	0.19	0.32	0.56
Model Training time, s	55.25	27.7	66.42	45.75	31.73

Two workers test executions with the top 5 RMSE values are aggregated in Table 12. The most precise results are shown for one worker (350 neurons, batch size 8, 50 epochs) with RMSE 3.06 (STDEV 1.01) and a model training time of 49.83 seconds. The predicted price change per cent is 0.68%.

Table 12: Elephas framework GOOG Alphabet Inc price prediction test run results as of 21.07.2022(350 neurons, two workers)

Number Of Workers	2	2	2	2	2
Batch Size	8	16	32	32	64
Number of Epochs	50	30	50	100	50
Change Percent	0.68	-1.33	-0.6	-2.05	0.78
MDAPE	1.74	1.33	2.32	2.51	2.28
RMSE	3.06	2.95	4.08	4.2	4.36
RMSE STDEV	1.01	0.63	1.24	1.4	1.22
Model Training time, s	49.83	24.71	26.89	44.8	22.93

Three workers test executions with the top 5 RMSE values are aggregated in Table 13. The most precise results are shown for one worker (350 neurons, batch size 32, 50 epochs) with RMSE 6.78 (STDEV 1.27) and a model training time of 49.83 seconds. The predicted price change per cent is -0.58%. Test combination with 100 epochs and three workers produced an error message during model training.

Table 13: Elephas framework GOOG Alphabet Inc price prediction test run results as of 21.07.2022(350 neurons, three workers)

Number Of Workers	3	3	3	3	3
Batch Size	8	16	32	32	64
Number of Epochs	30	30	30	50	50
Change Percent	-4.23	2.83	-1.23	-0.58	-0.25
MAE	7.07	8.11	8.32	5.88	6.14
RMSE	7.6	8.76	9.31	6.78	7.13
RMSE STDEV	6.35	5.74	2.3	1.27	2.81
Model Training time, s	33.44	23.81	19.95	26.34	22.93

Best accuracy results are received from one worker configuration. With an increasing number of workers distributed model training declining accuracy. Overall Elephas Framework is not mature yet for performing distributed model training.

6.4 Experiment 4. Keras Tuner Testing

Keras Tuner test results with the lowest RMSE of 3.06 (STDEV 0.33) are obtained on batch sizes 8, 100 epochs and 350 neurons per trial (Table 14). Keras tuner tests RMSE accuracy varies between 3.06 and 6.91. Also, test execution results in Table 14 show a maximum model training time of 4878.33 seconds (batch size 8, 100 epochs per trial), which indicates that it could not be used for critical data forecasting.

Table 14: GOOG Alphabet Inc price prediction test run results as of 22.07.2022 (Keras Tuner)

Experiment Name	Keras Tuner Local PC	Keras Tuner Local PC	Keras Tuner Local PC
Batch Size	8	8	8
Number of Neurons	350	350	350
Number of Epochs	30	50	100
MAE	2.52	2.49	2.5
RMSE	3.33	3.16	3.06
RMSE STDEV	0.49	0.56	0.33
Model Training time, s	1372.5	2283.55	4878.33

6.5 Discussion

1. Prediction accuracy and Running time best performers.

   Test results demonstrated that the best RMSE accuracy results are shown by PySpark-based code implementation both on Google Colab and Local PC. Best average RMSE value of 1.993 (standard deviation 0.006) is received on PySpark-based Colab implementation (100 epochs, 700 neurons, batch size 32). Overall best performance and accuracy tests are received on Google Colab Pro+ GPU and Local

PC platforms using GPU. This could be explained that TensorFlow GPU libraries are more efficient than the ones used with CPU or TPU. The lowest RMSE values are received on tests with hyperparameters having 350 or 700 neurons, 100 epochs and batch sizes 16 or 32. Tests executed with a batch size of 64 or 128, 30 epochs didn't show acceptable stable results. Google Colab TPU testing with distributed *TPUStrategy* model training shows better model training times over the non-distributed run, but RMSE accuracy in distributed training tests declines. Lower RMSE accuracy results are on a TPU-based test without using model training *TPUStrategy* but still didn't produce close accuracy similar to tests using GPU. Very interesting results are observed when comparing Pandas-based vs PySpark-based code executed in both distributed *MirroredStartegy* and non-distributed in Google ColabPro+ GPU. In distributed mode Pandas-based code mode training takes more time than in non-distributed mode, while PySpark-based code executes faster in distributed mode. Also, there is a big gap in model training time: Pandas-based test results (350 neurons, 100 epochs, batch size 32) 962.38 s vs 72.09 s in Colab GPU and 129.15 s in MacBook Pro with GPU. A similar timing difference appears in PySpark tests as well. AWS EMR model training time for the same hyperparameters (350 neurons, 100 epochs, batch size 32) is 313 seconds for Pandas-based code and 311 seconds for PySpark-based code. AWS Lambda tests show both performance and accuracy poor results and shouldn't be considered for intensive ML calculations, unless, AWS will introduce GPU processing and lift a 15 min processing time limit. AWS EMR tests demonstrate average accuracy and long processing time in comparison to GPU optimised environments. In some Elephas tests (one worker) it was noted RMSE 2.05 (Table 11), but it wasn't consistent.

2. Spark vs Pandas
   PySpark vs Pandas test results shows that there are no clear accuracy differences between PySpark or Pandas-based solutions. PySpark-based solution on GPU-based platforms tests model training time is longer than Pandas-based (Figure 7). But AWS EMR tests show, that there is not much difference between PySpark and Pandas-based code executions.
   Also, considering relatively small data sets, it shouldn't be a real difference between using Pandas Data frame and PySpark SQL application. It would increase performance on multi-million rows and a large number of column data sets to perform ETL data manipulation and preparation for model training. PySpark-based code is using Spark Application and Spark Context session to scale data processing and model training, which will help on big data sets. At the same time, PySpark SQL has a different syntax and limited functionality for the manipulation of data in comparison with Pandas.

3. Keras Tuner
   Keras Tuner tests show that this framework is not mature yet considering accuracy and model training time. It is a good tool for getting indicative results and experimental testing.

4. Elephas
   Elephas RDD model training demonstrated relatively good results only with one worker. With an increasing number of workers model accuracy is critically dropping and not acceptable for use. Distributed Sequential model training with multiple

workers is not well developed yet and should not be used with two or more workers.

5. Limitations
   Building Docker image for AWS Lambda required to have Intel-based PC, as TensorFlow officially is not supporting ARM-based solutions.
   During experiments due to access issues, it was not possible to configure the AWS EMR cluster to work from AWS SageMaker Studio. The same relates to AWS Fargate-based tests on college access.
   Google Colab Pro+ monthly subscription is quite high (50 Euro per month) and Google could ban GPU access due to fair policy usage.

6. Improvements
   To improve research it could be done by adding following elements:

   - Spark MLLib LSTM model training to perform end-to-end processing data, model training and predicting in Spark and compare performance and accuracy with TensorFlow using the same LSTM layers and hyperparameters.
   - A relatively small dataset using 5 years period of time is not big enough to test the PySpark SQL framework.
   - Include AWS SageMaker and Google GCP platforms for testing.

7 Conclusion and Future Work

This research paper investigates, tests and compares frameworks and technologies and contributes to Research Question:

> To what extent Machine Learning Cloud-Based Services can enhance the efficiency of stock price prediction and replace it to support investors and stock market traders?

Research work objectives are:

1. Research which platform is delivering the lowest RMSE and model training time when using Pandas-based code or PySpark-based code in distributed and non-distributed model training mode?

2. Research how accurate and fast Elephas RDD model training is.

3. Research how accurate and fast is Keras Tuner model training?

For delivering answers to these questions it was reviewed options to design, implement and execute ML solutions using Pandas-based code and PySpark-based code in both distributed and non-distributed model training mode, Keras Tuner and Elephas RDD to train a Time Series Sequential LSTM model on Google Colab, AWS EMR, AWS Lambda and Local PC. Jupyter Lab Notebooks are used on a programming basis.
ETL Jupyter Notebook has decoupled from the model the training part and completes end-to-end data retrieval, preparation, feature engineering and delivering the combined data set for the model training part of the project.
For completion of model training testing, it was prepared Jupyter Lab notebooks and deployed into Google Colab, AWS EMR, Jupyter Lab on local PC and AWS Lambda

docker image. Test results are aggregated into *results.csv* file and delivered for analysis and report.
The lowest RMSE values and model training time was observed on PySpark-based code executed on Google Colab GPU and Local PC with GPU. But Pandas-based code delivers similar RMSE accuracy as well. The fastest model training time belongs to Google Colab GPU. Distributed model training mode using *MirroredStrategy* works faster with PySpark-based code and slower with Pandas-based code. Tensorflow *TPUStrategy* distributed testing using Google Colab Pro+ TPU significantly improves model training time but delivers lower accuracy. Overall, test results show that on the GPU-based platform Pandas-based code training time is faster, with nearly similar to PySpark-based code accuracy. An increasing number of epochs, number of neurons and decreasing batch size consistently increase model training time on both PySpark and Pandas solutions.
Elephas distributed model training shows its efficiency and accuracy only on one worker configuration. It was implemented and tested only on the Google Colab Pro+ GPU environment. Local PC installation didn't succeed due to hardware limitations. Overall, the Elephas framework, especially in multi-worker mode is not mature enough for reliable model training.
Keras Tuner didn't deliver the best accuracy results and could be used only in the initial stages of ML model testing.
AWS Lambda usage for ML model training is limited due to its processing time limit. It is not fit to execute Tensorflow as fast as GPU solutions.
AWS EMR solution didn't deliver the best accuracy and model training time results. Too expensive to use.
Research summary. The main idea of this research is to establish the most efficient cloud-based solution delivering the best accuracy and model training time while performing Stock Price prediction task implemented on selected cloud-based platforms. During conducting research it was achieved following:

- Tested and proved the efficiency of Cloud-based GPU and TPU-distributed model training IT cloud infrastructure when performing ML LSTM Stock price prediction. Best accuracy and performance results are demonstrated by Google Colab Pro+ GPU solution with TensorFlow non-distributed model training.

- Tested and measured performance and accuracy of TensorFlow *MirroredStrategy*, *TPUStrategy* and Elephas RDD distributed model training when performing ML LSTM Stock price prediction. Proved the efficiency of using *TPUStrategy* distributed ML model training

- Tested and measured performance and accuracy of Cloud-based AWS EMR, AWS Lambda solutions. As per test results, these setups should be avoided when doing intensive ML model training computations.

- Tested and measured the performance and accuracy of model training using Google Colab Pro+ and Keras Tuner. As per the results, this tool needs more research and testing experiments and could not be recommended as reliable yet.

- Tested and measured PySpark vs Pandas data frame performance and accuracy. Both frameworks are delivering similar accuracy, but show different model training times. Proved that PySpark framework works faster with TensorFlow *MirroredStrategy* distributed ML model training.

Future work. As the PySpark-based solution showed good results on GPU-based platforms, it would be a good consideration to do the next step and implement the project using AWS Glue Spark-based server-less solutions for ETL job and keep data set in Spark from end-to-end processing. Potentially it is possible to improve the speed and quality of ML model training by experimenting with Spark ML library MLLib in a server-less GPU-based solution. Also, considering the Keras Tuner framework, results could be improved by using more hyperparameter search strategies. Distributed model training show interesting results. The next step would be enhancing research by using multi-GPU instances for distributed multi-worker model training.
Future work summary:

- Add experiments to measure and compare model training time and accuracy using Horovod and Tensorflow MultiworkerMirroredStrategy and MirroredStrategy multiple GPU distributed model training.
- Get a bigger dataset and scale ML model training with Spark ML Lib for use with PySpark.
- To simplify the test execution process, a harness workflow tool could be developed for automating the model training hyperparameter iterations by providing either hyperparameters range or particular values.

8 Acknowledgements

I would like to say thank you for my family's support while they were missing my attention during lectures and research work. Also, I would like to express my gratitude to Dr Horacio Gonzalez-Velez for guiding to accomplish research project.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[2] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: A system for Large-Scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. Savannah, GA: USENIX Association, Nov. 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi

[3] J. Verbraeken, M. Wolting, J. Katzy, J. Kloppenburg, T. Verbelen, and J. S. Rellermeyer, "A Survey on Distributed Machine Learning," *ACM Computing Surveys*, vol. 53, no. 2, pp. 1–33, Mar. 2021. [Online]. Available: https://dl.acm.org/doi/10.1145/3377454

[4] T. O. Hodson, "Root-mean-square error (RMSE) or mean absolute error (MAE): when to use them or not," *Geoscientific Model Development*, vol. 15, pp. 5481–5487, Jul. 2022, publisher: Copernicus Publications. [Online]. Available: https://search.ebscohost.com/login.aspx?direct=true&AuthType=ip,cookie,shib&db=edsdoj&AN=edsdoj.bc6508e8a630422e9731b3d704f6d940&site=eds-live&scope=site&custid=ncirlib

[5] "Project Jupyter | Home," last accessed 2022-08-01. [Online]. Available: https://jupyter.org/

[6] "Google Colab," last accessed 25/07/2022. [Online]. Available: https://colab.research.google.com/signup

[7] "Big Data Platform – Amazon EMR – Amazon Web Services," last accessed 25/07/2022. [Online]. Available: https://aws.amazon.com/emr/?nc1=h_ls

[8] "Serverless Computing - AWS Lambda - Amazon Web Services," last accessed 25/07/2022. [Online]. Available: https://aws.amazon.com/lambda/

[9] R. Philipp, A. Mladenow, C. Strauss, and A. Völz, "Machine Learning as a Service: Challenges in Research and Applications," in *Proceedings of the 22nd International Conference on Information Integration and Web-based Applications & Services*. Chiang Mai Thailand: ACM, Nov. 2020, pp. 396–406. [Online]. Available: https://dl.acm.org/doi/10.1145/3428757.3429152

[10] E. Hesamifard, H. Takabi, and M. Ghasemi, "Deep Neural Networks Classification over Encrypted Data," in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*. NY, USA: Association for Computing Machinery, 2019, pp. 97–108. [Online]. Available: https://doi.org/10.1145/3292006.3300044

[11] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service," *Proceedings on Privacy Enhancing Technologies*, vol. 2018, no. 3, pp. 123–142, 2018.

[12] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, "Chiron: Privacy-preserving machine learning as a service," *CoRR*, vol. abs/1803.05961, 2018. [Online]. Available: http://arxiv.org/abs/1803.05961

[13] M. Kesarwani, B. Mukhoty, V. Arya, and S. Mehta, "Model Extraction Warning in MLaaS Paradigm," in *Proceedings of the 34th Annual Computer Security Applications Conference*, ser. ACSAC '18. NY, USA: Association for Computing Machinery, 2018, pp. 371–380. [Online]. Available: https://doi.org/10.1145/3274694.3274740

[14] "What is distributed training? - Azure Machine Learning | Microsoft Docs," last accessed 2022-08-01. [Online]. Available: https://docs.microsoft.com/en-us/azure/machine-learning/concept-distributed-training

[15] A. Sergeev and M. D. Balso, "Horovod: fast and easy distributed deep learning in tensorflow," *ArXiv*, vol. abs/1802.05799, 2018.

[16] "Module: tf.distribute | TensorFlow Core v2.9.1," last accessed 2022-08-12. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/distribute

[17] M. Pumperla, "Elephas: Distributed Deep Learning with Keras & Spark," Jul. 2022, original-date: 2015-08-13T12:09:19Z. [Online]. Available: https://github.com/maxpumperla/elephas

[18] J. Sen and T. D. Chaudhuri, "Stock price prediction using machine learning and deep learning frameworks," in *Proceedings of the 6th International Conference on Business Analytics and Intelligence, Bangalore, India*, 2018, pp. 20–22.

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[20] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural Computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[21] D. M. Q. Nelson, A. C. M. Pereira, and R. A. de Oliveira, "Stock market's price movement prediction with lstm neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1419–1426.

[22] "The Sequential model | TensorFlow Core," last accessed 2022-08-01. [Online]. Available: https://www.tensorflow.org/guide/keras/sequential_model

[23] H. Abbasimehr and R. Paki, "Improving time series forecasting using LSTM and attention models," *Journal of Ambient Intelligence and Humanized Computing*, vol. 13, no. 1, pp. 673–691, Jan. 2022. [Online]. Available: https://link.springer.com/10.1007/s12652-020-02761-x

[24] A. Sagheer and M. Kotb, "Time series forecasting of petroleum production using deep LSTM recurrent networks," *Neurocomputing*, vol. 323, pp. 203–213, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231218311639

[25] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi *et al.*, "Keras-tuner," https://github.com/keras-team/keras-tuner, 2019.

[26] C. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Proceedings of the international AAAI conference on web and social media*, vol. 8, no. 1, 2014, pp. 216–225.

[27] R. A. Mendoza Urdiales, A. García-Medina, and J. A. Nuñez Mora, "Measuring information flux between social media and stock prices with Transfer Entropy," *PLOS ONE*, vol. 16, no. 9, p. e0257686, Sep. 2021. [Online]. Available: https://dx.plos.org/10.1371/journal.pone.0257686

[28] "PySpark Documentation — PySpark 3.3.0 documentation," last accessed 2022-08-01. [Online]. Available: https://spark.apache.org/docs/latest/api/python/

[29] "Module: tf | TensorFlow Core v2.9.1," last accessed 2022-08-12. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf

[30] "Use Cases, Tutorials, & Documentation | Twitter Developer Platform," last accessed 2022-08-01. [Online]. Available: https://developer.twitter.com/en

[31] "Yahoo Finance - Stock Market Live, Quotes, Business & Finance News," last accessed 2022-08-01. [Online]. Available: https://finance.yahoo.com/?guccounter=1

[32] "Stock Market Prediction using Multivariate Time Series Models in Python," last accessed 25/07/2022. [Online]. Available: https://www.relataly.com/stock-market-prediction-using-multivariate-time-series-in-python/1815/