Performance Analysis of Spark on AIRS Cloud Using HiBench Benchmarks and AI Training Ability Evaluation Using AIBench

Changwen LI
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
118010134@link.cuhk.edu.cn

Minghong XIA
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
118010340@link.cuhk.edu.cn

Hanyang XIA
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
118010339@link.cuhk.edu.cn

Ziqi GAO
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
118010077@link.cuhk.edu.cn

Dongyang WU
School of Data Science
The Chinese University of Hong Kong, Shenzhen
Shenzhen, China
118010324@link.cuhk.edu.cn

***Abstract*—In this paper, we evaluate the performance of AIRS Cloud platform. Four cloud benchmarks were tested on AIRS Cloud Platform: Random Forest (RF), Gradient Boosted Tree (GBT), Text-to-Text and Text Summarization. Cloud benchmarking results are analyzed with the speed, throughput, speed up, efficiency. We will focus on the speed, throughput, resource usage, and the accuracy of the models for those four tasks. Also, we will introduce how the data set and hyper parameters affect the training process of the neural networks, and the performance of the model.**

Keywords—HiBench, AIBench, Spark, AIStation, Cloud Computing, AIRS

I. Introduction

Cloud computing and artificial intelligence are getting gaining more and more attention and popularity. It is of great significance to do experiments to test the performance of the AIaaS platform. RF and GBT are executed on HiBench with 4 core Xeon processors, 400 GB disks and 8GB RAM. The execution processes and results analysis of RF and GBT are very similar. Therefore, RF and GBT are written together in one section. Text to Text and Text Summarization are two benchmarks from two AIBench benchmarks loaded on AIRS' AIStation with 2 CPU cores and 1 GTX2080Ti GPU. Those two benchmarks will be introduced separately in detail in the second and third part of the paper.

II. HiBench Benchmarks: Classification Using Random Forest and Gradient Boosted Tree

A. *Introduction*

Both Random Forest and Gradient Boosted Tree are well-known classification method based on decision trees. Random Forest (RF) is one of the popular machine learning method. After invented by Leo Breiman and Adele Cutler, RF becomes well-known as it could help to avoid the bias-variance tradeoff of decision trees [1]. A “random forest”, as its name indicates, is made up of various “decision trees”. As a supervised machine learning model, a random forest learns to map data to outputs in the training phase of model building. The algorithm could be divided into 4 processes [2]: 1. Create an equal-size dataset by bootstrapping; 2. When current decision tree is to split, randomly choose a few properties (far less than all properties) and choose the most useful one as the property of split; 3. Repeat 1-2 for multiple times, thus multiple decision trees are created; 4. On prediction, decision “vote” for the answer.

Comparing with a single decision tree, Random Forest has at least 2 advantages. First, comparing with a deep decision tree which consumes a lot of time to run, an RF could run multiple decision trees simultaneously, as the decision trees in the forest are mutually irrelevant, which provides a great prospect for parallel computing. Second, a complex decision tree could make the model of high variance, which will lead to overfitting; in contrast, a simple decision tree could make the model of high bias. RF could offset the effect of a “special” factor, which could help to avoid the bias-variance tradeoff.

Instead of using the whole data set as training materials like ordinary decision trees, RF will create an equal-size dataset by sampling in original dataset with replacement. Such process will be repeat multiple times to create multiple different decision trees. When predicting data, different decision trees will give different answers and the final results will be the average (or by “voting”).

Gradient boosted tree (“GBT” for short) is another popular machine learning method. It constructs an additive regression model, utilizing decision trees as the weak learners [3]. After being invented by Friedman, GBT is well-known for its high adaptability and accuracy [4]. The GBT first start with a leaf of the average value. Second, create a decision tree based on the prediction errors. Then, scale the tree’s contribution to the final prediction according to its learning rate. Repeat second step until the maximum number of trees is met. The GBT shares some similarities with the RF but it is far more complicated. They both use multiple trees to predict the results. However, RF creates tree by random attributes, while GBT obtains the new tree through previous prediction errors.

The RF and GBT algorithm are widely used classification algorithms, which makes them acceptable indicators to measure the performance of a computing platform. In this project, RF and GBT are tested and executed in spark, which is loaded on AIRS cloud computing system. This workload is implemented in *spark.mllib* and the input data set is generated by *RandomForestDataGenerator.scala* and *GradientBoostedTreeDataGenerator* [5].

B. *Configuration and Testing Procedures*

In this project, the size of data and the partition number and the shuffle partition number are controlled by modifing the file of hibench.conf in directory conf. It offers the configuration to both Spark and Hadoop. The path of data input is reconfigured before start running experiments. The size of data input is also modified in this configuration file. Another important configuration file is spark.conf which is also under the directory conf. The number of executors and cores in YARN can be manipulated in spark.conf. The numbers could be controlled by a script written in Python, which could be found in the appendix.

C. *Experimental Results*

Two tests are implemented in the experiments of RF and GBT: the first test is implemented with modified data sets, while the second test is implemented with modified number of partitions and shuffle partitions. The corresponding performance matrices are calculated and analyzed.

a). Tests on Different Data Sets: In these tests, different input data sets are tested. There are 6 data sets which are denoted as tiny, small, large, huge, gigantic and bigdata in both RF tests and GBT tests. Their corresponding values are around 10 KB, 400 KB, 7.7 MB, 14.9 GB, 22.4 GB and 32.8 GB for test of RF, and 9.578 KB, 397.230 KB, 15.316 MB, 30.575 MB, 61.093 MB and 91.610 MB for test of GBT. The *hibench.yarn.executor.num* and *hibench.yarn.executor.cores* are both set to be 4. The *hibench.default.map.parallelism* and *hibench.default.shuffle.parallelism* is altered. The corresponding duration time and throughput are recorded. The results are shown in the following graphs and tables.

The relationship between duration time and data input size of RF is shown in table 1, and the relationship between duration time and data input size of RF is shown in table 2. The duration time is obtained from the *hibench.report* in */home/team5/github/HiBench/report*, meaning the total time of an execution. When the data size becomes large enough, the duration time increases as the size of the data input increases. To better estimate the relationship, figure 1 and 2 is drawn whose data unit is "bytes" for RF and GBT.

Input Data Size (Byte)	Average Duration Time (s)
35,201,204,268	1760.474071
24,000,602,305	1321.035125
16,000,602,272	661.7051875
8,059,543.5	33.2331875
406,831.5	30.96973438
9,785.38	30.43571

TABLE 1: RELATIONSHIP BETWEEN AVG. DURATION TIME AND AVG. INPUT DATA SIZE FOR RF

Data input size (bytes)	Duration time (s)
9808	31.424
406,764	47.439
16,060,500	147.897
32,060,500	222.705
64,060,500	416.679
96,060,500	491.21

TABLE 2: RELATIONSHIP BETWEEN AVG. DURATION TIME AND INPUT DATA SIZE FOR GBT

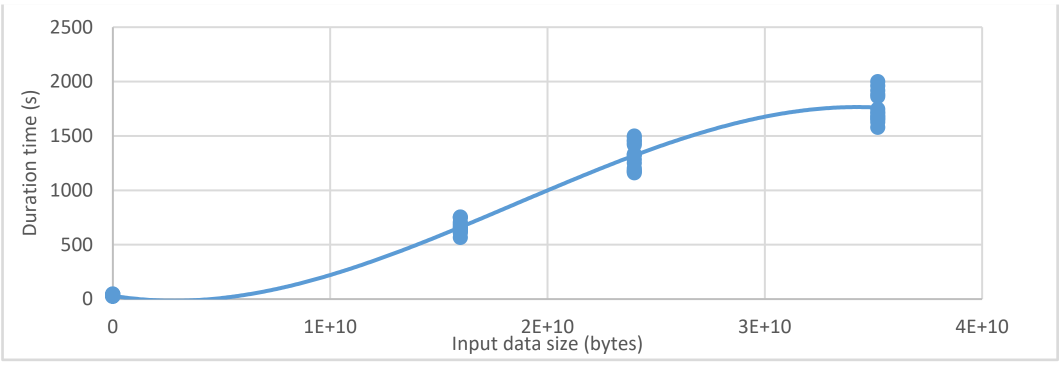


FIGURE 1: RELATIONSHIP BETWEEN DURATION TIME AND INPUT DATA SIZE FOR RF

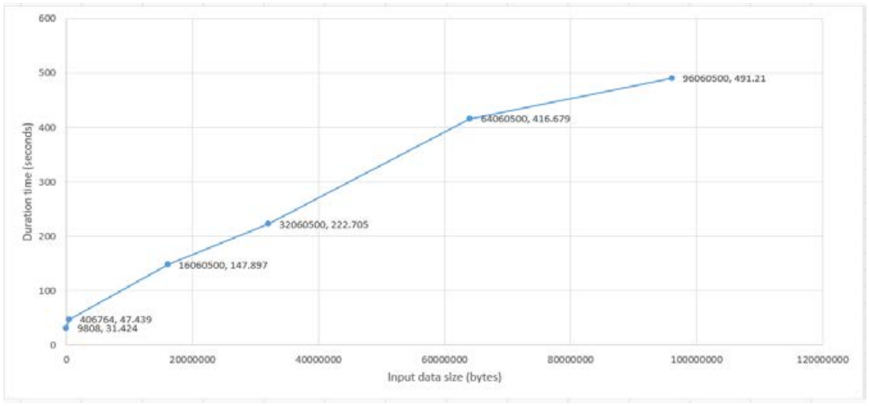


FIGURE 2: RELATIONSHIP BETWEEN DURATION TIME AND INPUT DATA SIZE FOR GBT

It is clear that the curves in both figure 1 and figure 2 are approximately linear except for some exceptions. This phenomenon is intuitive in figure 1 and 2, where most of the data scale is tested for multiple times, and they are all displayed in the figure. However, such phenomenon will no longer exist when the data size is not large enough:

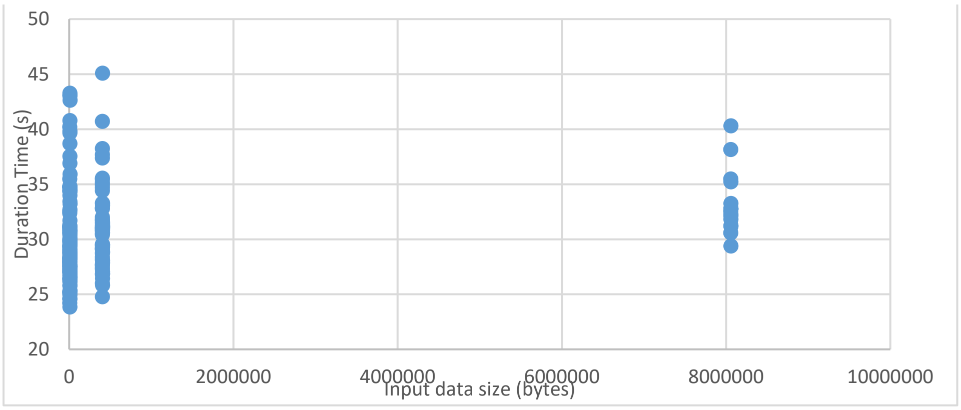


FIGURE 3: RELATIONSHIP BETWEEN DURATION TIME AND INPUT DATA SIZE FOR RF (PARTIAL)

Figure 3 only displays data with scale “tiny”, “small” and “large”. It is obvious that when data size is not big enough, most of the duration time will be 35±10s.

The relationship between data throughput and data input size is shown in following tables and figures. The results of RF tests are shown in table 3 and figure 4, the results of GBT tests are shown in table 4 and figure 5. When the input data size switches from tiny to small, from small to large, and from large to huge, the increase of throughput is linear. In order to obtain an intuitive relation between two factors in scales, the figure displayed is log-log plot. Both the x coordinate (data input size) and y coordinate (throughput) are replaced by their logarithmic values. Thus, the figure 4 and figure 5 were drawn after the throughput is transferred to its logarithm. When the input data size switches from huge to gigantic and from gigantic to bigdata, the increase of throughput is unstable.

Average Throughput (byte/s)	Input Data Size (Byte)
19,995,298.33	35,201,204,268
18,168,027.37	24,000,602,305
24,180,862.68	16,000,602,272
242,514.9107	8,059,543.5
13,136.422	406,831.5
321.5098317	9,785.38

TABLE 3: RELATIONSHIP BETWEEN AVG. THROUGHPUT AND AVG. INPUT DATA SIZE FOR RF

Data input size (bytes)	Throughput (bytes/s)	Throughput/node (bytes/s)
9808	312	78

406764	8574	2143
16060500	108592	27148
32060500	143959	35989
64060500	153740	38435
96060500	195558	48889

TABLE 4: RELATIONSHIP BETWEEN THROUGHPUT, THROUGHPUT/NODE AND INPUT DATA SIZE FOR GBT

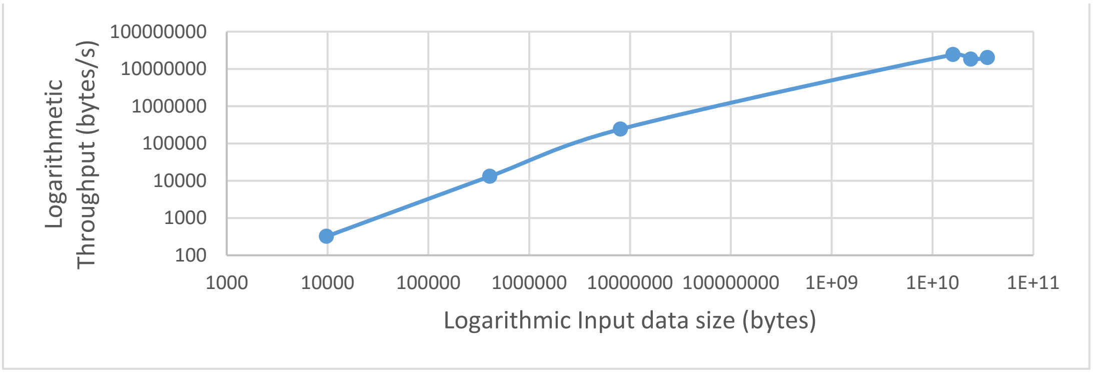


FIGURE 4: RELATIONSHIP BETWEEN THROUGHPUT AND INPUT DATA SIZE FOR RF (AFTER LOGARITHM)

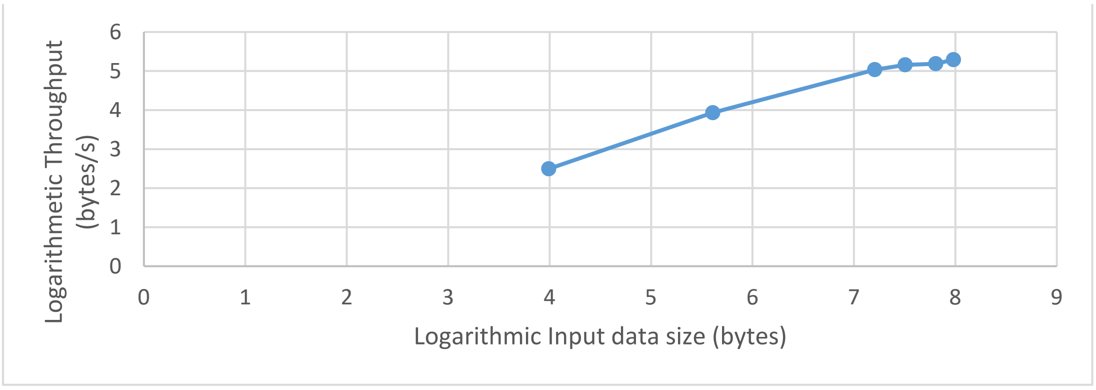


FIGURE 5: RELATIONSHIP BETWEEN THROUGHPUT AND INPUT DATA SIZE FOR GBT (AFTER LOGARITHM)

b). Tests on different number of partitions and shuffle partitions: In these tests, different number of map parallelism and shuffle parallelism are altered. The corresponding duration time is recorded. The input data size is set as *gigantic* (14.9 GB) in RF tests and *large* (15.316 MB) in GBT tests.

Either map parallelism or shuffle parallelism is altered in a test. It is hard to control the execution time of parallel tasks to a constant value while it is fairly easy to control the workload unchanged. As a result, Amdahl's law instead of Gustafson's law is implemented to calculate the scaled speedup. The form of Amdahl's law is shown below in following equation:

$$Speedup(\Lambda) = \frac{T(1)}{T(\Lambda)} \qquad (1)$$

The speed up here means the speed gain due to the use of increasing number of computing nodes on the AIRS cloud platform instead of AWS EC2. The original $T(1)$ is the running time of a certain application running on a 1-ECU instance [6], while here $T(1)$ is the running time of GBT running on one single core, which is 1496.904s in RF test and 349.052s in GBT tests. T(Λ) represents the running time on the cluster. The efficiency is calculated as well. The formula used to calculate the efficiency is shown below:

$$Efficiency(\Lambda) = \frac{Speedup(\Lambda)}{\Lambda} \qquad (2)$$

The meaning of $N(\Lambda)$ is different from its original meaning. Its original meaning is the number of ECU instances, while here it indicates the quantity of cores in AIRS cloud platform.

Firstly, the shuffle partition number is set to be 5 and 6 in RF tests and GBT tests, respectively. The relationship between number of partitions, duration time, speed up, efficiency is shown in Table 5. While the similar data for GBT is recorded in table 6.

Number of partitions	Duration time (s)	Speed-up	Efficiency
4	1159.116	1.255984733	0.313996183
3	1168.918	1.245452632	0.311363158
2	1331.902	1.093047386	0.273261847
1	1416.933	1.027452956	0.256863239

TABLE 5: RELATIONSHIP BETWEEN NUMBER OF PARTITIONS, SPEED UP AND EFFICIENCY OF RF

Number of partitions	Duration time (s)	Speed-up	Efficiency
1	290.818	1.200242076	0.300060519
6	171.115	2.039867925	0.509966981
11	206.241	1.692447186	0.423111796
16	171.367	2.036868242	0.50921706
21	172.139	2.027733401	0.50693335

TABLE 6: RELATIONSHIP BETWEEN NUMBER OF PARTITIONS, SPEED UP AND EFFICIENCY OF GBT

Using the same method and setting the partition number to be 4 and 3 in RF tests and GBT tests, respectively, the relationship between number of shuffle partitions, duration time, speed up, efficiency is shown in Table 7. The similar data is recorded in table 8.

Number of shuffle partitions	Duration time (s)	Speed-up	Efficiency
4	1183.611	1.229991948	0.307497987
3	1159.116	1.255984733	0.313996183
2	1202.404	1.210767762	0.30269194
1	1174.083	1.239973665	0.309993416

TABLE 7: RELATIONSHIP BETWEEN NUMBER OF SHUFFLE PARTITIONS, SPEED UP AND EFFICIENCY OF RF

Number of shuffle partitions	Duration time (s)	Speed-up	Efficiency
1	170.014	2.053077982	0.513269495
3	162.346	2.150049893	0.537512473
5	183.97	1.897331087	0.474332772
7	174.107	2.004813132	0.501203283
9	177.38	1.967820498	0.491955125
11	155.496	2.244765139	0.561191285

TABLE 8: RELATIONSHIP BETWEEN NUMBER OF SHUFFLE PARTITIONS, SPEED UP AND EFFICIENCY OF GBT

D. Analysis

Three relationships will be analyzed in this report: between test accuracy and data input size, between data throughput and data input size, and between number of (shuffle) partitions and duration time.

a). Relationship between Test Accuracy and Data Input Size.

The test error could be retrieved from */home/team5/github/HiBench/report/rf/spark/bench.log*, the figure showing the test error is attached in the appendix. Because the analysis processes for RF and GBT in this section are same. Therefore, for the limitation of article length, here only the analysis of Random Forest is shown.

It is obvious that Test Accuracy = 1 - Test Error. The relationship between test accuracy and data input size is shown in following tables and figures. In order to obtain an intuitive relation between two factors in scales, the figure displayed is log-log plot.

Number of shuffle partitions	Examples	Test Error
Bigdata	20000	0.029059108
Gigantic	10000	0.030643742
Huge	10000	0.029492219
Large	1000	0.063551828
Small	100	0.161192472
Tiny	10	0.7

TABLE 9: RELATIONSHIP BETWEEN AVG. ERROR AND INPUT DATA SCALE FOR RF

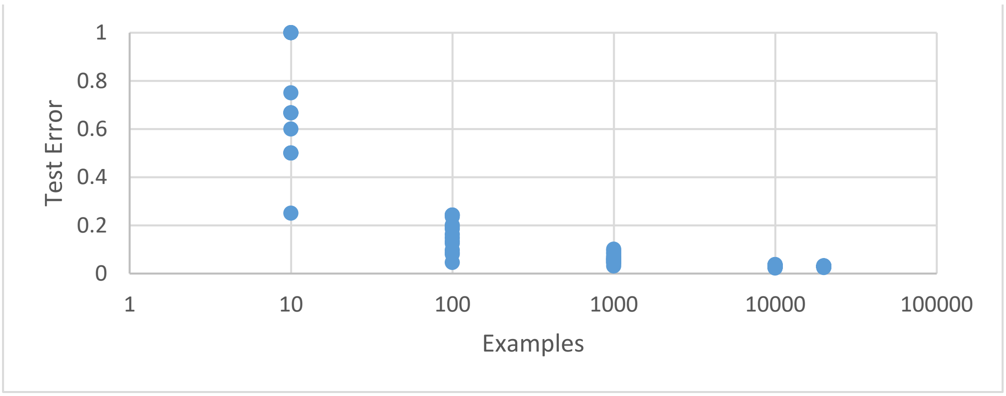


FIGURE 6: RELATIONSHIP BETWEEN ERROR AND INPUT DATA SIZE FOR RF

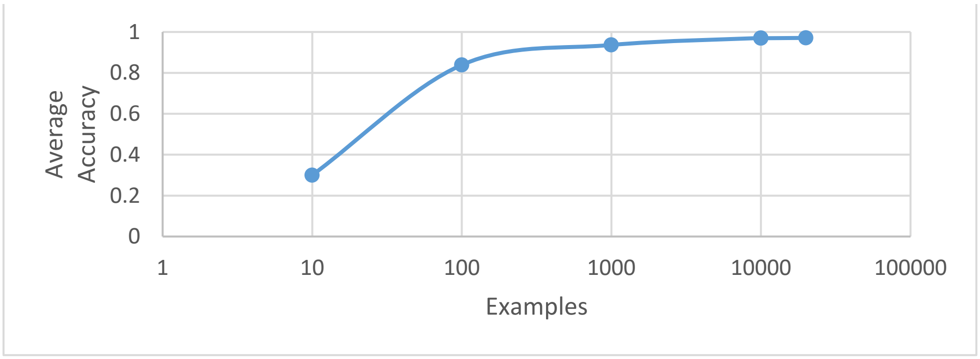


FIGURE 7: RELATIONSHIP BETWEEN AVG. ACCURACY AND INPUT DATA SIZE FOR RF

It could be seen from Figure 6 that along with increasing input data scale, both bias and variance of RF will decrease. Meanwhile, the curve in Figure 7 becomes more flattened when input data size becomes larger.

The curve in Figure 6 could be recognized as Logistic curve:

It is assumed that when input data size is only 1, the accuracy of RF will reach 0. Therefore, we obtained the following table:

Examples	Average Accuracy
20000	0.970940892
10000	0.969356258
10000	0.970507781
1000	0.936448172
100	0.838807528
10	0.3
1	0

TABLE 10: AUGMENTED RELATIONSHIP BETWEEN AVG. ERROR AND INPUT DATA SCALE FOR RF

If a curve could be recognized as Logistics curve, it should fit the following format:

$$f(x) = \frac{a}{1+e^{-bx}} + c \Leftrightarrow \frac{a}{f(x)-c} = 1 + e^{-bx} \tag{3}$$

It is obvious that the accuracy of RF ranges from 0 to 1, and will approach 1 when the input data size is infinite. Therefore, we map the range [0,1] to [0.5,1], which is the range of Logistics curve. The table will be as follows:

Examples	Modified Value
20000	0.985470446
10000	0.984678129
10000	0.985253891
1000	0.968224086
100	0.919403764
10	0.65
1	0.5

TABLE 11: RELATIONSHIP BETWEEN MODIFIED VALUE AND INPUT DATA SCALE FOR RF

The modified data should fit the following format:

$$\frac{1}{f(x)} = 1 + e^{-bx} \tag{4}$$

Thus, we use the reciprocal of modified value minus 1 to create a table, and the result shows the exponential relation.

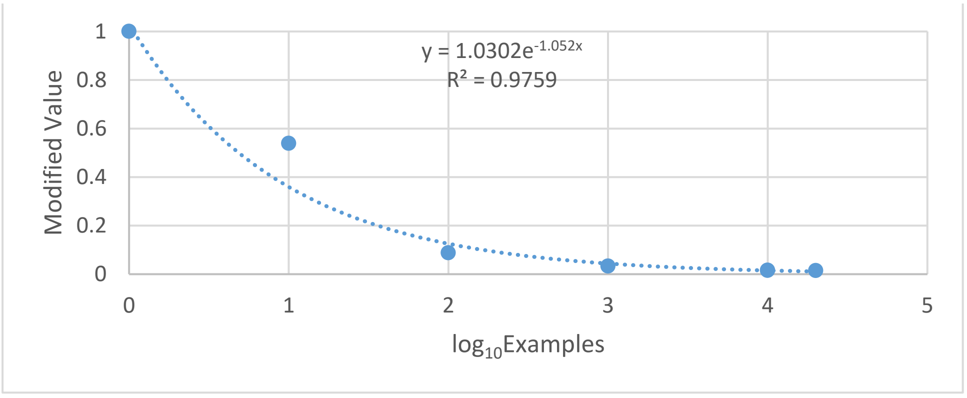


FIGURE 8: RELATIONSHIP BETWEEN MODIFIED VALUE AND INPUT DATA SCALE (AFTER LOGARITHM) FOR RF

Therefore, the Logistics curve is proved.

b). Relationship between Data Throughput and Data Input Size

When data input size is small, the relationship between data throughput and data input size could be considered linear. A possible reason for the linear increase is that when data input size is not large enough, the communication time between CPU cores are far more than computing time of CPU. Therefore, when data input size increase, the computation of CPU is not saturated, thus the time will hardly increase. As a result, the throughput increases. In contrast, when data input size is large enough, the throughout will keep stable.

c). Relationship between Number of (Shuffle) Partitions and Duration Time

There seems to be no significant correlation between the number of partitions (or shuffle partitions) and the duration time. This phenomenon is counter-intuitive because the computing speed should increase as the number of partitions increase. The reason for this phenomenon is that the increasing number of partitions does not really increase the parallelism. While the maximum number of available cores is 4, when the number of partitions is larger than 4, the program is not completely parallel. Thus, the increase of partitions number does not have a significant influence on duration time.

d). Relationship between Input Data Size and Duration Time

The curve between duration time and data input size is approximately linear because of the Spark application execution mechanism. When an application is loaded onto the spark framework, the driver node (DAG Scheduler, Task Set Manager, etc.) distributes and schedules it into several partitions. Each partition will be executed on one core. Therefore, since the number of partitions and cores are the same, the execution time is proportional to input data size. However, there is one exception occurs. For GBT, when the data set is tiny (9.578 KB), the data point cannot fit in the linear curve. The major cause of this phenomenon is that, the duration time is more than parallel computing time. It also includes other time that is irrelevant to input data size, such as initialization and configuration time. For instance, time spent on transferring python application code into spark jobs by *ContextHandler* is irrelevant to data size. According to the *bench.log*, in the test with tiny dataset (9.578 KB), it takes 15 seconds to first activate *DAGScheduler*, while it takes 14 seconds in the test with bigdata input (91.610 MB). However, their total duration time varies a lot (31 seconds and 491 seconds). When the data size is small, these sorts of time occupy higher percentage of total duration time, which makes it unable to fit in a linear curve.

III. Text-to-text Translation Based on Aibench

A. Introduction

The Text-to-Text translation used transformer model. The core idea behind the Transformer model is self-attention—the ability to attend to different positions of the input sequence to compute a representation of that sequence [7]. Transformer utilize Scaled dot product attention and Multi-head attention and can handles variable-sized input using stacks of self-attention layers instead of RNNs or CNNs. For the architecture, most competitive neural sequence transduction models have an encoder-decoder structure. The encoder maps an input sequence of symbol representations (x1, ..., xn) to a sequence of continuous representations z = (z1, ..., zn). Given z, the decoder then generates an output sequence (y1, ..., ym) of symbols one element at a time. At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next. This Transformer follows this overall architecture using stacked self-attention and point-wise, fully. connected layers for both the encoder and decoder [7]. The detail explanation of encoder-decoder structure and attention can be seen in appendix.

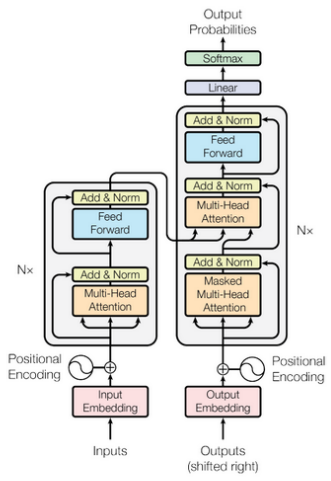


Figure 9: The transformer architecture

B. Configuration and Testing Procedures

The configuration and Testing Procedures includes the understanding of epoch, batch size and other key parameters, reading the result of preprocess procedure, train procedure and transform. Please see in the appendix for detail explanation.

C. Output and analysis of the program

a). Tests on Different Data Sets: The program is trained on the dataset of English and German language provided by this course. It is trained to convert English to German and German to English. Also, a new dataset called IWSLT'15 English-Vietnamese data was also utilized to convert English to Vietnamese and Vietnamese to English from paper *Stanford Neural Machine Translation Systems for Spoken Language Domains*[8].

a1). Test on German to English.

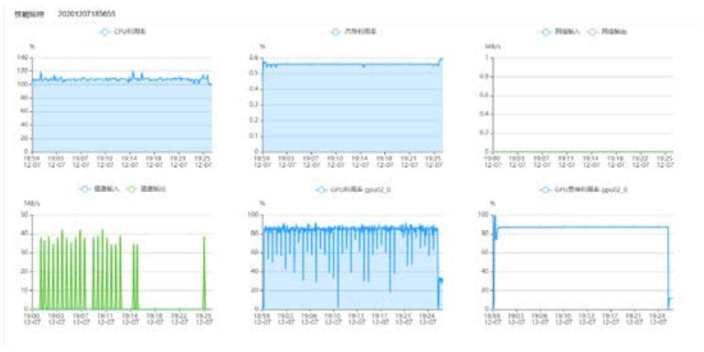


Figure 10: output of the CPU/GPU using rate, memory using rate and the input/output of the network/disk

In the 26 minutes training, the using rate of CPU is around 108%, the using rate of memory is about 56%, the output of the network is almost 0, the using rate of the GPU is about 85%, the using rate of GPU memory is about 85%. The disk will have output of 30 to 45 MB/s about every minutes to the store the current trained model.

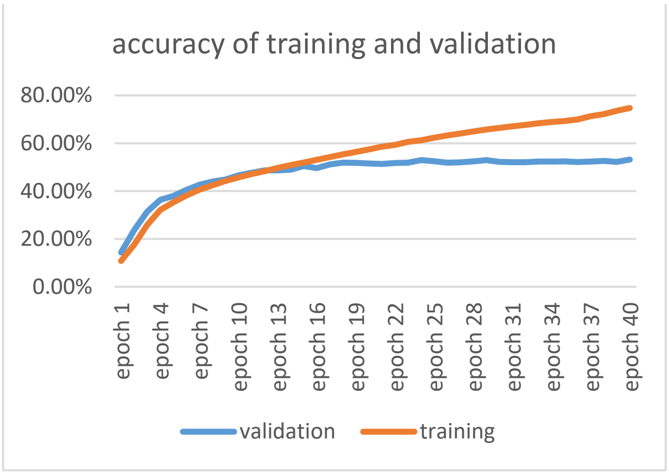


Figure 11: the accuracy of training and validation

The accuracy of training begins from 10.0% to 75%, the accuracy of validation begins from 13% to 52%. After the 15th epoch, the validation accuracy tend to go up and down without the trend of going up. However, the training accuracy keeps going up. In the next page, we will increase the epoch to 100 to find out the complete accuracy of training and validation.

We also use our trained model which has validation accuracy of 52% and training accuracy 87% to translate the test.de file, and below is the output. The quality is acceptable for such a short time of training (40 minutes). Below showed the translated result of test.de:

man in an orange hat is working with an orange hat </s>

a white dog is running in front of a white fence. </s>

a girl in a <unk> sweatshirt is holding a stick with a stick </s>

five people wearing <unk> are standing outdoors with <unk> in the background. </s>

people are sitting outside of a house. </s>

a man in a blue dress and a woman in a dress of a group of people in <unk> </s>

a group of people standing outside a store shop. </s>

a boy in a red jersey tries to catch the red ball while another boy watches. </s>

a guy working on a building. </s>

a man in a vest sits on a chair holding a <unk> </s>

a mom and son enjoying a nice day on a nice day. </s>

man <unk> as the man <unk> a football <unk> holding his hands <unk> </s>

a woman in a kitchen holding food of food. </s>

a2). Test on English to German.

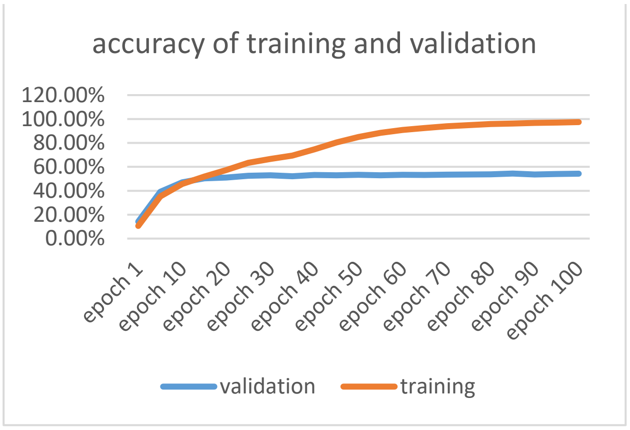


Figure 12: the accuracy of training and validation of German to English with epoch 100

In epoch 100, it is obvious that the training accuracy keeps going up to 99%, however the validation accuracy remain about 58% after 30 epoch. There are some possible reasons to explain. First, the distribution of validation set and training set may not be the same. Second, the default validation set is too small, smaller than one twentieth of the training set, which may cause the not satisfying performance of the model. Third, the transformer model itself may have some limitation of performance instead of blaming for the validation set.

a3). Test on English to Vietnamese and Vietnamese to English

After completing the default dataset offer by in this course we want to train our model in other dataset to test its applicability. We got this dataset from https://nlp.stanford.edu/projects/nmt/ [9]. It is called IWSLT'15 English-Vietnamese data [9]. The training dataset contained train.en and train.vi, which has size of 133K. As the limitation of training time, we only choose 15000 lines for train.en and train.vi. We chose 5000 lines for the validation in the test set of Stanford in 2012 and 2013 of 2718 lines and another 3318 lines from train.en and train.vi(There is no overlapping between training set and validation set). The difficulty of training increase as some sentences has length that longer than 50, it will affect the performance.

a3.1). English to Vietnamese

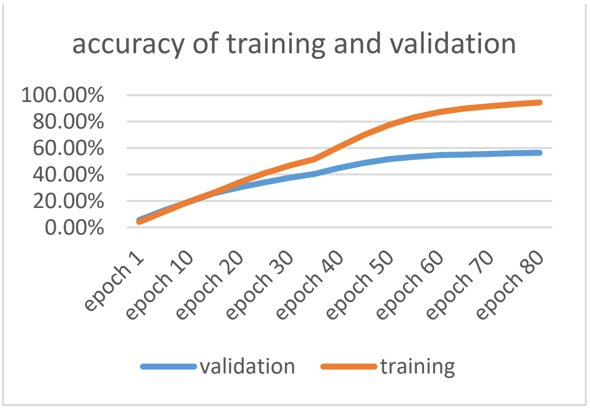


Figure 13: the accuracy of training and validation of English to Vietnamese

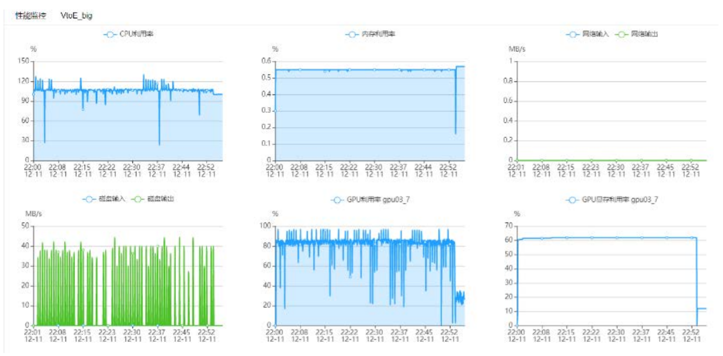


Figure 14: output of the CPU/GPU using rate, memory using rate and the input/output of the network/disk

From the picture, we could find that the training and validation and testing took 51 minutes for 80 epoch in total. The CPU usage rate is about 105%, and the GPU usage rate is about 85%, the memory usage rate is about 55%, the disk will have output of 30 to 45 MB/s about every one minute. The output of disk is the written of model. The usage rate of GPU memory is about 62%.

b). test on other settings

Here the number of epochs, batch size, and size of training and validation data is concerned. They all affect the performance.

Firstly, for the **batch size**, the larger batch set will have better performance, however, if we choose a very large batch size like 512 or 1024, it will cause error. As the program has to deal with data of batch size parallelly. The CUDA will run out of memory. This is common in deep learning and every developer should be aware of. We test different batch size in the German to English training task.

Batch size	Training time (s)	Final validation accuracy (epoch 40)	Final Training accuracy (epoch 40)
320	1575	52.75 %	72.457%
256	1652	52.394%	73.46 %
128	1811	52.052 %	75.65 %
64	2402	13.125 %	17.229 %
32	4397	13.125 %	17.033 %

Table 12. comparison between different batch size of data in German to English with epoch 40

With the increase of batch size, the training time decrease. In the epoch 40, the epoch size 128,256 and 320 gave satisfying result while size 32 and 64 offered unsatisfying result.

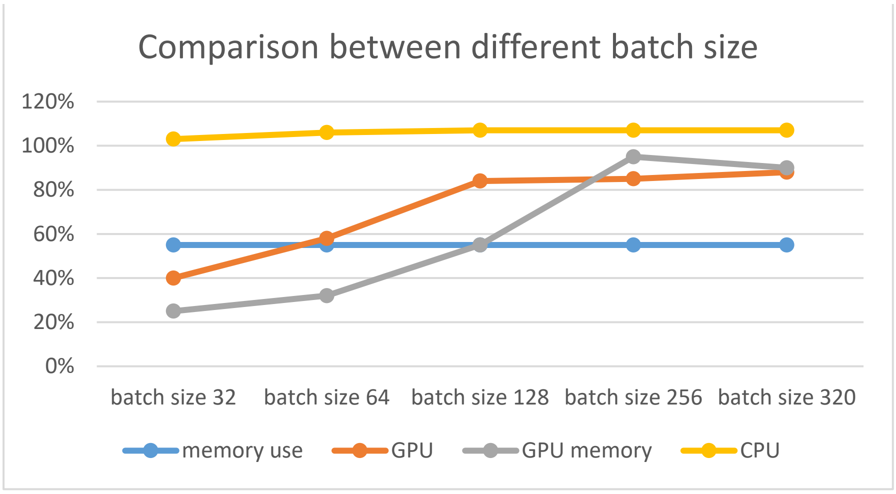


Figure 15: memory use, GPU use, GPU memory use and CPU use of different size of batch

With the increasing of the batch size from 32 to 320, the memory use did not vary too much, it may because the training data size is not very large. For the GPU memory, it increase from 25% to about 95% from batch size of 32 to batch size of 256 and go down a little from size 256 to 320. For the GPU usage, it increased sharply from 40% to 80% from the batch size 32 to 128 and did not vary much from size 128 to size 320. For the CPU, it increases slowly from 103% to 107% from the batch size 32 to batch size 128, and remain about 107% from batch size 128 to batch size 320.

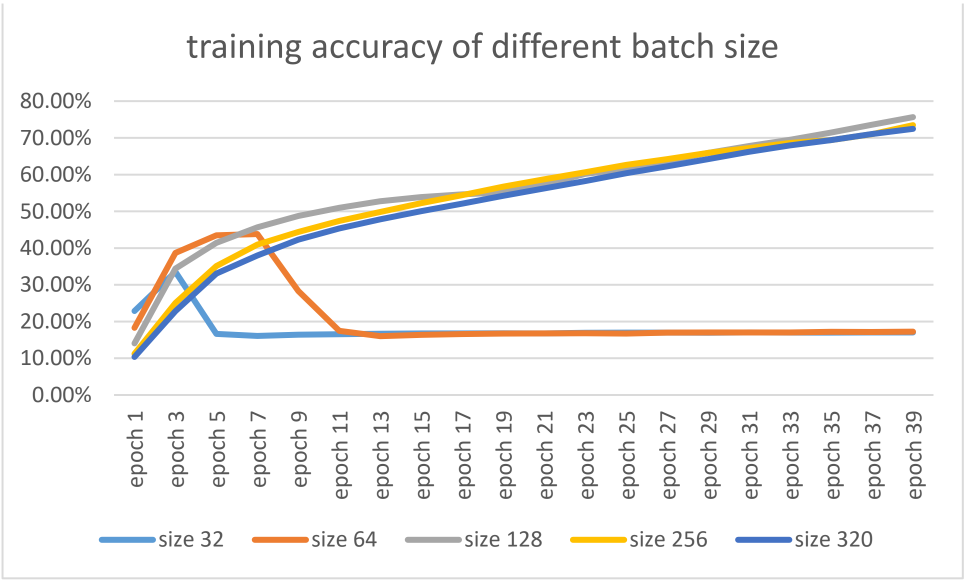


Figure 16: training accuracy between different size of batch

Here we record the training accuracy from batch size 32 to batch size 320 in epoch 1 to 40. For the batch size of 32, it increased sharply from epoch 1 to epoch 3 and then decreased sharply from epoch 3 to epoch 5 and remain stable at the accuracy about 17%. The case of batch size of 64 is similar to the batch size 32 case. For the batch size of 128, 256 and 320, it is another story.

They both increase gradually from epoch 1 to epoch 39 and achieve the final accuracy about 75%. However, as we only test the epoch of 40, it is not sure that which size of epoch will have the best performance. But for the accuracy in epoch 40, batch size of 128 is the best. If we take time into consideration, both epoch 128, 256 and 512 are acceptable.

Secondly, for the **number of epochs**, it is clear that for the epoch that less than 100, the accuracy of training will go after epoch 40 while the accuracy of validation will remain stable after epoch 30. It is meaningless to run the epoch larger than 100. In real practice, we should balance between training time and accuracy. 100 epochs of training took 66 minutes and 40 epochs took 27 minutes with the same of other setting. The training time is not the most significant in this project. However, in real case, when people use this model to do the translation, a balance between accuracy and training time should be highly considered.

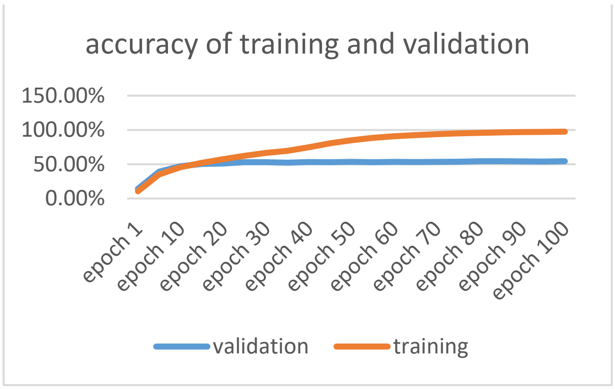


Figure 17: output of English to German with epoch 100

Thirdly, for the **size of training data and validation data**, the effects to the performance is complex. It is clear that the training data and validation data should have similar distribution to achieve good performance.

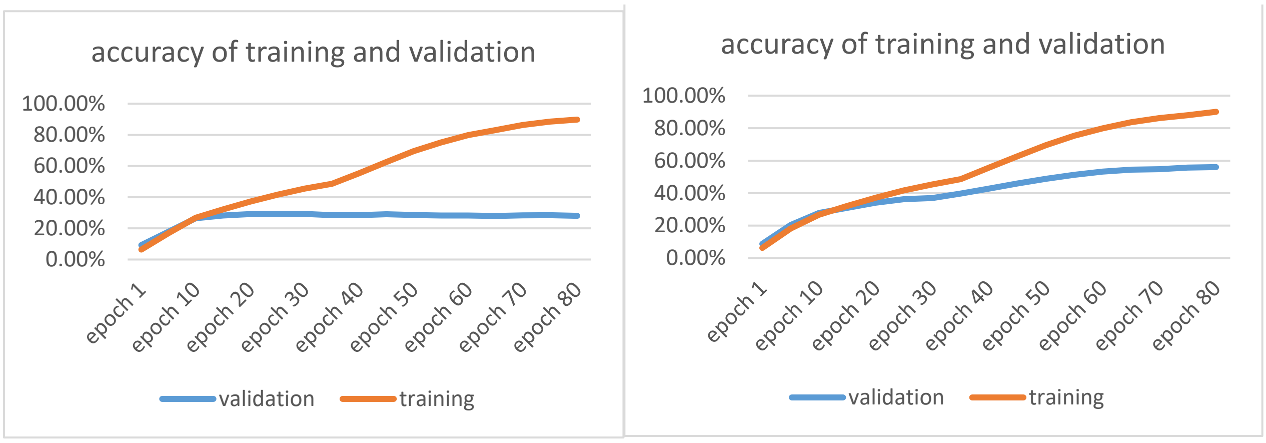


Figure 18: output of Vietnamese to English with different training dataset. The left picture is with small 7500 lines of training set and Stanford 2012 dataset and 2013 dataset of 2821 lines for validation. The right picture is with 15000 lines of training set and combination of Stanford 2012 and 2013 testing set and 3318 lines of Stanford dataset as validation set.

In figure 18, the accuracy of validation didn't perform well with small validation set. The validation accuracy is only about 25% in epoch 80. When the training set is enlarged 15000 lines and the validation set is added with 3318 lines of the Stanford IWSLT'15 English-Vietnamese training data (there is no overlapping in the set of training set and validation set), the accuracy of validation increase to 58% in epoch 80. There is two possible reasons for the result, the first is that the increase of size of training data contribute to the applicability of the model, the second is that the Stanford 2012 and Stanford 2013 test case did not have the similar distribution with the training dataset provided by Stanford. However, when we use the 3318 lines of Stanford training data as validation set and use the previous 15000 lines data as training set, the validation accuracy is only 25% in 80th epoch. This makes the analysis really complex.

D. Discussion

Firstly, GPU is essential in deep learning, without the support of GPU, it would calculate at a very low speed.

Secondly, the parameter of training played an important role in the performance. For example, the number of epochs, the batch size, the choice of loss function and the layer and size of the network all effect the performance.

Thirdly, the performance we may pay attention to accuracy, training time and resource usage. The accuracy is the things that we care most in this assignment. For the training time, it is vital in industry, but for this assignment, we don't consider it as the most vital issue. For the resource usage, it really matters in this assignment. We have to admit that the resource is limited as each group only has one GPU and two CPU, which is not enough to calculate the speedup, parallel computing is also not doable. Even though we could calculate with one or two CPU, the difference of performance is not significant and seems meaningless. As the memory for GPU is limited, if we choose the batch size to be 512 or 1024 or larger, the program will run out of the memory.

Fourthly, it is interesting to run the program on different dataset. We run in the dataset that convert between English and Vietnamese. The output of the accuracy of the training and validation showed that the program suffer from over fitting. This may because the training dataset and validation dataset have different distribution.

E. Further Improvement

As time limited, we only test English to German and German to English, and the convention between English and Vietnamese. If we have chances, we will do the following things: Firstly, utilize different model from Stanford NLP dataset. Secondly, we could use many GPU and CPU to do the parallel computing and calculate the speedup and elasticity. Thirdly, in the deep learning area, utilize different methods to improve the performance, such as data enhancement, solving the problem of over fitting, solve of the problem of gradient descent and so on.

IV. TEXT SUMMARIZATION ON AIBENCH

A. Introduction

This benchmark task is to generate the text summary, which is important for search results preview, headline generation, and keyword discovery. This task can be naturally cast as mapping an input sequence of words in a source document to a target sequence called summary.

The description of this benchmark is shown in the table below. After the training, the computer can generate a text summary of any article in English.

No.	Component Benchmark	Algorithm	Data Set	Software Stack
DC-AI-C14	Text Summarization	Sequence to sequence Model	Gigaword Dataset	PyTorch + Tensorflow

TABLE 13. Detail of This Benchmark [13]

The benchmark is run on one node of AIRS Cloud AIaaS Pool and the detail of the hardware of the tested node is as follows:

Component	Spec
Intel Xeon-6230 CPU	2 Cores of the CPU (2.0 GHz per core)
RTX 2080Ti × 1	13.4 TFlops with 11GB GDDR6 Graphics Memory[15]

TABLE 14. Hardware Detail of The Tested Node [14]

The training is conducted on the AIStation Platform developed by Inspur, which can rapidly deploy a training environment for deep learning, and comprehensively manage deep learning training task. [16]

This benchmark focuses on the accuracy of the training model, the training time, the GPU usage, which are major industry concerns. All of the data needed are recorded by the AIStation automatically.

B. Models

The model used in this benchmark is a sequence to sequence (RNN-based) encoder-decoder abstractive model [13]. We use the adjective 'abstractive' to denote a summary that is not merely a selection of a few existing passages or sentences extracted from the source [18]. Also, unlike the RNN used in the text translation, the summary generated for the article is typically concise. It does not depend very much on the length of the source in summarization, and the relation between the source text and the summary is not obvious.

Another concern of the text summarization is how to handle out-of-vocabulary (OOV) words. OOV happens when the model meets a word that is unseen or rare during the training [18].

As a result, to solve the problems mentioned, we need a more powerful model. The model used in this benchmark adds two 'components' to the standard RNN model:

a. Supervised learning combined with reinforcement learning [19]: Reinforcement training can give the regular training a prediction to make the summaries more readable.

b. Hybrid Pointer-generator Network [13]: To solve the OOV problem, we have a switch to control the decoder of the model. When the switch is turned on, the decoder produces a word from its target vocabulary in the normal fashion. However, if the switch is turned off, the decoder instead generates a pointer to one of the word-positions in the source [18].

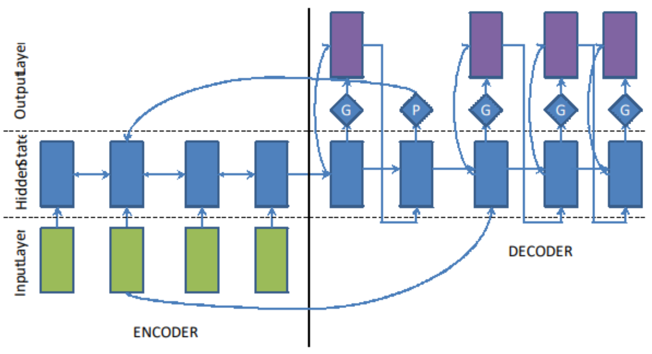


Figure 19. Sequence to Sequence Model with Pointer [17]

C. About the dataset:

Here we use the Gigaword generated from the OpenNMT [20]. The data size of the data is about one gigabyte with the articles and the corresponding titles. We can divide the data into two parts. About 95% of the input is taken as the training data set, while the last 5% is used for validation. The data files are saved in the data/unfinished directory. Because the model can only read the binary file, we need to combine the title and the article together. An example part of the input binary file is enclosed in the appendix 12. During the preprocessing, those files are also divided into 4MB chunks. Finally, we have 254 training chunks and 13 valid chunks with one test batch. Each chunk have no more than 15,000 examples (Each example means one article/title).

File	Size
train.article.txt	662MB
train.title.txt	189MB
valid.article.filter.txt	33MB
valid.title.filter.txt	9.58MB

TABLE 15. Training Data's Size[14]

Then, let us have a brief look at the training data. Most of the training article is about 150 words, and most of the title is about 50 words.The length of each example is as follows:

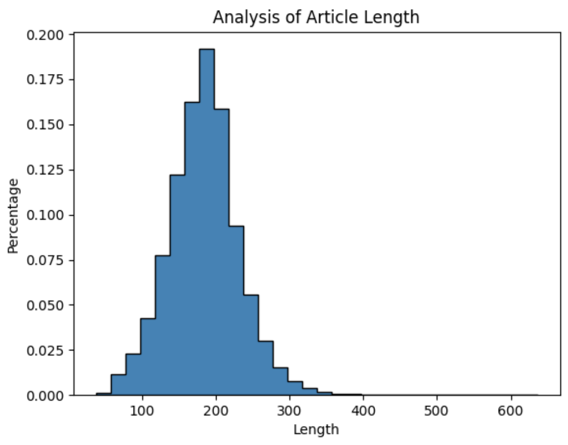


Figure 20. Historgram of the Length of the Training Article

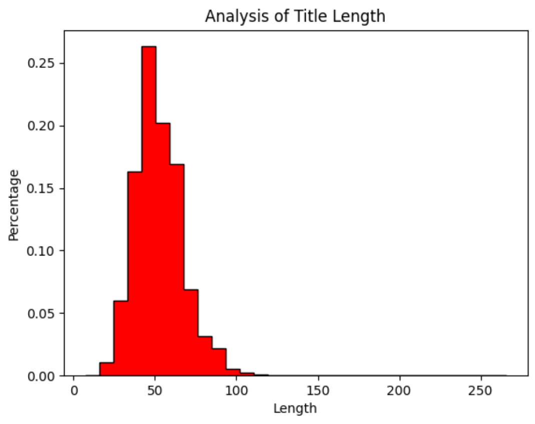


Figure 21. Historgram of the Length of the Training Abstract

As a result, we can conjecture that our model worked best when the article is about 150 words and can generate a title with approximately 50 words.

D. Model Training without reinforcement learning:

Here we check the training time, accuracy, and GPU usage of the training. We use the total overall loss of the maximum likelihood estimation to estimate the accuracy of the model during the training.

$$P_{vocab} = softmax(V'(V[s_t, h_t^*] + b) + b')$$

where $V', V, b, and\ b'$ are learnable parameters, while h^t is the context vector which can be read from the encoder. S_t is the decoder's state which decides whether we need to use the words from the vocabulary table or need to extract a word from the

source when OOV happens. Finally, we can get a probability distribution over all words in the vocabulary table. Then, when OOV not occurs, we can have the final distribution from which to predict the words w in summary:$P_{vocab}(w) = P(w)$
Then we calculate the loss of each timestamp:

$$loss_t = -\log P(w_t^*)$$

w_t^* is the target word in that timestep, and the overall loss for the whole sequence is the mean of the losses:

$$loss = \frac{1}{T}\sum_{t=0}^{T} loss_t \; [17]$$

We use this loss to measure the accuracy of the training model. Here we first check the effect of the vocabulary size, and the batch size used here is set to 400:

a. Vocabulary Size and MLE_Loss

In this exeperiment, we will first check whether the vocabulary size will have any effect on the training accuracy measured by MLE_Loss:

According to the chart driven from the training log, we can conclude that the vocabulary size should not make a great difference in training accuracy.

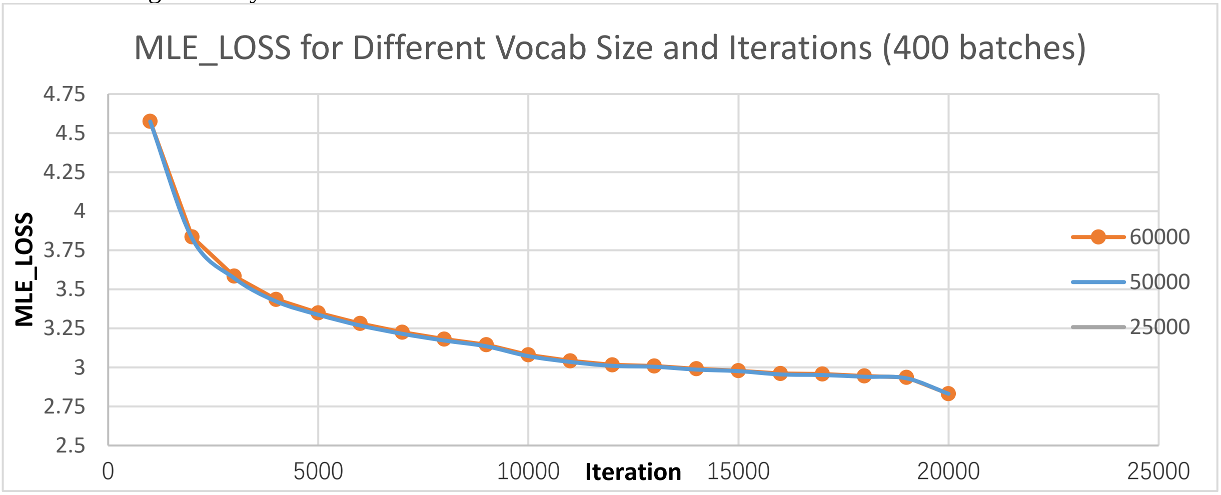


Figure 22. MLE_LOSS for Iterations and Vocabulary Sizes (batch size 400)

We think this phenomenon is reasonable because during the training, we will not come across the OOV problem and the calculation of the MLE loss is also irrelevant to the vocabulary size.

b. Vocabulary Size and Source Usage:

Since the time for this project is limited, and training a very accurate model that performs well in the validation set needs about 65000 iterations ($\approx 7\ epcohes$) (please check the appendix 14 for more information), we only train 20000 iterations ($\approx 2\ epoches$) for each vocabulary size model. Since we have known that the time spent on every 5000 iterations is almost the same, we appro2ximately calculate the total training time using the following formula:

$$T_{65000} = \frac{65000}{20000} \times T_{20000}$$

Vocabulary Size	Training Time (20000 Iterations)	Derived Training Time (65000 Iterations)
60,000	**About 7 hours (7h03min)**	**22.75 hours**
50,000	**About 6 hours (5h51min)**	**19.5 hours**
25,000	**About 5.5 hours (5h40min)**	**19.25 hours**

TABLE 16. Training Time for Different Vocabulary Size Settings

Vocabulary Size	CPU Usage (2 Cores)	GPU Usage	GPU Memory Usage
60,000	**About 170%**	**About 50%**	**95%**
50,000	**About 160%**	**< 50 %**	**93%**
25,000	**About 170%**	**< 40 %**	**66%**

TABLE 17. Resource Usage for Different Vocabulary Size Settings

This table shows us that we can get a shorter training time by reducing the vocabulary size. Also, memory usage and gpu usage will also drop when we reduce the vocabulary table size. However, the CPU is not affected by vocabulary size. The reason is that we use CUDA to control our GPU to train the model instead of CPU.

E. Model Training with reinforcement learning:

In this experiment, we will test the effect of the batch size using the reinforcement learning method. The vocabulary size for the following experiemnt is set to 25,000.

Batch size in text summarization means how many tasks will articles and titles be propagated through the network[21]. As a result, if we set the batch size to 400, we train 400 examples per iteration. The batch size should make a huge difference to the GPU usage for we need to calculate 400 examples together. The following chart shows how the batch size affects the usage of GPU during the reinforcement learning:

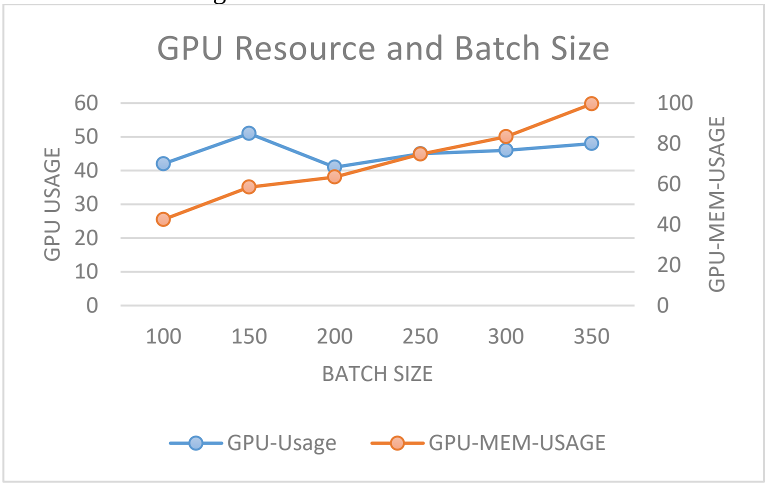


Figure 23. GPU Resource Usage for Different Batch Size Settings

We can learn from the chart that: As the batch size improves, the memory usage improves. When the batch size is 350, we almost run up the GPU memory. Also, if the batch size is about 400, the training will run out of the graphic memory of this node's GPU and fail to train the model.

As to the GPU Usage, we use the peek value here because of the fluctuation, and we can see that there is no obvious relationship between the usage and the batch size.

F. Evaluation of Models:

According to the essays that we have read, the most popular way to measure the quality of the text summarization model is Rouge value[17]. Rouge stands for Recall-Oriented Understudy for Gisting Evaluation[22]. It works by matching the overlap of n-grams of the generated and reference summary. The value of Rouge can be calculated by the following formula:

$$\text{ROUGE-N} = \frac{\sum_{S \in \{Referemce\ Summaries\}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in \{Reference\ Summaries\}} \sum_{gram_n \in S} Count(gram_n)}$$ [23]

Also, Rogue-L = $\frac{LCS(X,Y)}{m}$, where LCS stands for longest common subsequence and m means the total length of the article[23].

Here we will compare the effect using Rouge value of all the three training methods (MLE Training, MLE + RL Training, RL Training) given by the developer. We conduct the reinforcement learning based on the model trained by MLE for 10,000 iterations. Other parameters are listed in the following table:

Batch_size	Vocabulary Size	Iteration ($\approx 2\ Epoches$)
200	25,000	20,000

TABLE 18. Hyper Parameters Setting of This Experiment

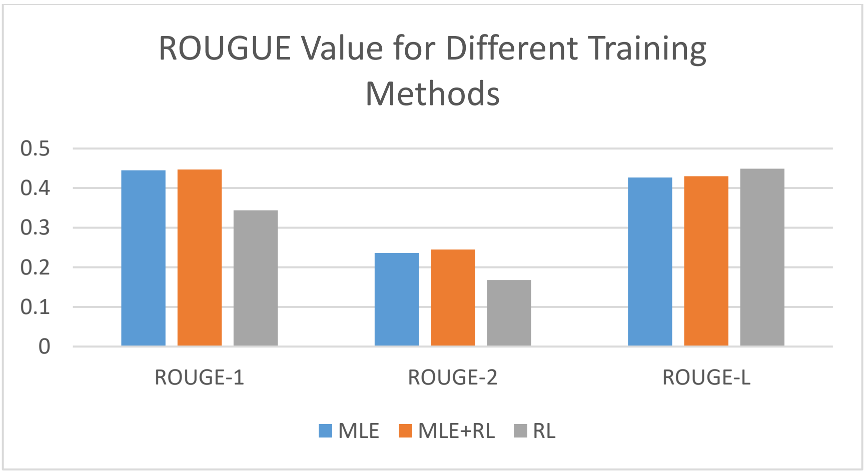


Figure 24. Rouge Value for Different Learning Methods

The data is got from the log of running the code in appendix 10. We take the average for both the 'f', 'p' and 'r' value in the log as the final rouge value. The chart above shows that MLE + RL have the best Rouge-1 and Rouge-2 value. RL method has the best Rouge-L value. Then, we can read the summarize generated by those methods. Please have a look at the appendix 18. In our group's opinion, we think the model trained by only MLE works the best. It does not show any repeated word, and the shows more information than the titles summarized by other models. However, all of them do not work well. We think we need more training iterations to improve the quality. Also, the vocabulary size of 25,000 is not enough. We still can see lots of unknown word (OOV occurs).

G. Further Work:

We did not test the effect of the batch size on the training time and the model's accuracy due to the limited time. Suppose we need to trian the model for 4 epochs. It is hard to speculate the training time because when we are training with the same amount of data, we make a trade-off between the number of iterations and the batches. Although the number of iterations is reduced, we need to spend more time on the forward propagation and backward propagation in each iteration in the neural network. Also, we can anticipate a lower number of iterations needed to reach a good MLE_Loss when the batch size is larger due to the features of gradient descend.

As to the model quality, we think that a larger batch size may negatively affect the quality of the model. We conjecture this point because the larger batch method will lead to the lack of generalization ability and make us hard to decide the value of the inherent noise in the gradient estimation. We need more experiments to prove our prediction.

H. Conclusion:

The performance of the node shows that this cloud node is qualified for the training of text summarization with the batch size not larger than 400. The bottleneck of this cloud node is the GPU memory. When we train the model with reinforcement learning using 400 batch size, the memory will be used up, and the train will fail. In contrast, the GPU's calculation resource is excessive. According to the experiment results, the GPU usage is always less than 50% during the whole process of this benchmark.

About the text summarization model: We think the rouge value is not that reasonable. We can check the summary from the model (please check the appendix). Although the rouge value is not bad, the summary is kind of confusing and broken. Also, this model cannot deal with numbers. All the training data and the valid data use # to replace of the number, which is also not satisfying.

V. Conclusion

In project 2, different kinds of experiments are executed. During the execution, performance of AIRS Cloud platform and performance of different deep learning training methods are evaluated.

From our experiments, we can conclude that our node on the AIRS cloud platform is qualified for most of the scenarios. We can conduct our experiments successfully without meeting big troubles caused by hardware. Only one thing that is not that satisfying is the GPU's memory. We will run out of the memory when conducting the reinforcement learning using the default hyper parameters given by AIBench.

Also, we need to conduct more experiments to prove our predictions. For example, how the batch size affects the accuracy and training time of the model mentioned in the last part.

VI. Appendix

1. Python script to configure and execute experiments on HiBench (Random Forest): auto.py

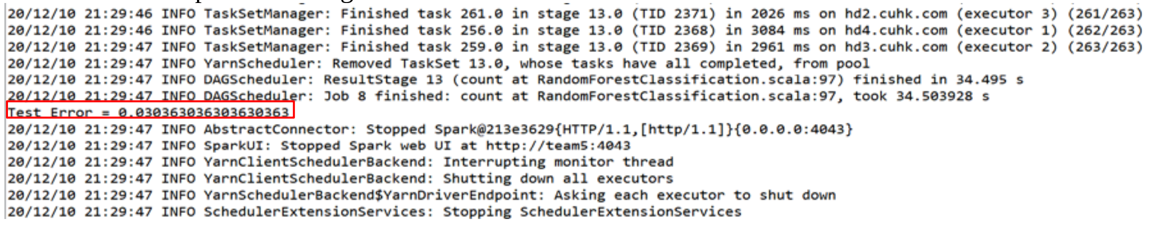
```python
import os
def cut(x1,x2):
    pp=""
    with open("/home/team5/github/HiBench/conf/hibench.conf") as f1:
        lines = f1.readlines()
    f2 = open("/home/team5/github/HiBench/conf/temp.conf","w")
    for i in range(len(lines)):
        st = ""
    if(lines[i].startswith("hibench.scale.profile")):
            t = lines[i].split()
            pp = t[1]
            st = lines[i]
    elif(lines[i].startswith("hibench.default.map.parallelism")):
            st = "hibench.default.map.parallelism         "+str(x1)+"\n"
    elif(lines[i].startswith("hibench.default.shuffle.parallelism")):
            st = "hibench.default.shuffle.parallelism     "+str(x2)+"\n"
        else:
            st = lines[i]
        f2.write(st)
    return pp
def cutsize():
    with open("/home/team5/github/HiBench/conf/hibench.conf") as f1:
        lines = f1.readlines()
    f2 = open("/home/team5/github/HiBench/conf/temp.conf","w")
    pp = ""
    for i in range(len(lines)):
        st = ""
        q = ""
        if(lines[i].startswith("hibench.scale.profile")):
            t = lines[i].split()
            if(t[1]=="tiny"):
                q = "small"
            if(t[1]=="small"):
                q = "large"
            if(t[1]=="large"):
                q = "huge"
            if(t[1]=="huge"):
                q = "gigantic"
            if(t[1]=="gigantic"):
                q = "bigdata"
            if(t[1]=="bigdata"):
                exit()
            st = "hibench.scale.profile                   "+q+"\n"
            pp = q
        else:
            st = lines[i]
        f2.write(st)
    return pp
def paste():
    with open("/home/team5/github/HiBench/conf/temp.conf") as f1:
        lines = f1.readlines()
    f2 = open("/home/team5/github/HiBench/conf/hibench.conf","w")
    for i in range(len(lines)):
        f2.write(lines[i])
n = 0
x1 = 4
x2 = 4
```

```
while(True):
    while(x1 > 0):
        while(x2 > 0):
            s = cut(x1,x2)
            paste()
            com1 = "cp -
R /home/team5/github/HiBench/report/rf/prepare /home/team5/github/HiBench/report/rf/reports/" + s + str(x1) + "-
" + str(x2) + "p"
            com2 = "cp -
R /home/team5/github/HiBench/report/rf/spark /home/team5/github/HiBench/report/rf/reports/" + s + str(x1) + "-
" + str(x2) + "s"
            os.system("sh /home/team5/github/HiBench/bin/workloads/ml/rf/prepare/prepare.sh")
            os.system(com1)
            os.system("sh /home/team5/github/HiBench/bin/workloads/ml/rf/spark/run.sh")
            os.system(com2)
            x2 -= 1
            #if x2 <=4 and (s == "bigdata" or s == "gigantic"):
            #    break
        x2 = 4
        x1 -= 1
        #if x1 <=4 and (s == "bigdata" or s == "gigantic"):
        #    break
    x1 = 4
    x2 = 4
    #cutsize()
    #paste()
    break
```

2. Python script to configure and execute experiments on HiBench (Gradient Boosted Tree): autoG.py

```
import os
def cut(x1,x2):
    pp=""
    with open("/home/team5/github/HiBench/conf/hibench.conf") as f1:
        lines = f1.readlines()
    f2 = open("/home/team5/github/HiBench/conf/temp.conf","w")
    for i in range(len(lines)):
        st = ""
    if(lines[i].startswith("hibench.scale.profile")):
            t = lines[i].split()
            pp = t[1]
            st = lines[i]
    elif(lines[i].startswith("hibench.default.map.parallelism")):
            st = "hibench.default.map.parallelism         "+str(x1)+"\n"
    elif(lines[i].startswith("hibench.default.shuffle.parallelism")):
            st = "hibench.default.shuffle.parallelism     "+str(x2)+"\n"
        else:
            st = lines[i]
        f2.write(st)
    return pp
def paste():
    with open("/home/team5/github/HiBench/conf/temp.conf") as f1:
        lines = f1.readlines()
    f2 = open("/home/team5/github/HiBench/conf/hibench.conf","w")
    for i in range(len(lines)):
        f2.write(lines[i])
x1 = 0
x2 = 0
while(x2 < 5):
    s = cut(3,5*x2+1)
    paste()
    com1 = "cp -
R /home/team5/github/HiBench/report/gbt/prepare /home/team5/github/HiBench/report/gbt/reports/" + s + str(x1) + "-
" + str(x2) + "p"
    com2 = "cp -
R /home/team5/github/HiBench/report/gbt/spark /home/team5/github/HiBench/report/gbt/reports/" + s + str(x1) + "-
" + str(x2) + "s"
    os.system("sh /home/team5/github/HiBench/bin/workloads/ml/gbt/prepare/prepare.sh")
    os.system(com1)
    os.system("sh /home/team5/github/HiBench/bin/workloads/ml/gbt/spark/run.sh")
    os.system(com2)
    x2 += 1
while(x1 < 5):
    s = cut(2*x1+1,5)
    paste()
    com1 = "cp -
R /home/team5/github/HiBench/report/gbt/prepare /home/team5/github/HiBench/report/gbt/reports/" + s + str(x1) + "-
" + str(x2) + "p"
    com2 = "cp -
R /home/team5/github/HiBench/report/gbt/spark /home/team5/github/HiBench/report/gbt/reports/" + s + str(x1) + "-
" + str(x2) + "s"
    os.system("sh /home/team5/github/HiBench/bin/workloads/ml/gbt/prepare/prepare.sh")
    os.system(com1)
    os.system("sh /home/team5/github/HiBench/bin/workloads/ml/gbt/spark/run.sh")
    os.system(com2)
    x1 += 1
```

3. Test errors in Spark bench.log



```
20/12/10 21:29:46 INFO TaskSetManager: Finished task 261.0 in stage 13.0 (TID 2371) in 2026 ms on hd2.cuhk.com (executor 3) (261/263)
20/12/10 21:29:46 INFO TaskSetManager: Finished task 256.0 in stage 13.0 (TID 2368) in 3084 ms on hd4.cuhk.com (executor 1) (262/263)
20/12/10 21:29:47 INFO TaskSetManager: Finished task 259.0 in stage 13.0 (TID 2369) in 2961 ms on hd3.cuhk.com (executor 2) (263/263)
20/12/10 21:29:47 INFO YarnScheduler: Removed TaskSet 13.0, whose tasks have all completed, from pool
20/12/10 21:29:47 INFO DAGScheduler: ResultStage 13 (count at RandomForestClassification.scala:97) finished in 34.495 s
20/12/10 21:29:47 INFO DAGScheduler: Job 8 finished: count at RandomForestClassification.scala:97, took 34.503928 s
Test Error = 0.030363036303630363
20/12/10 21:29:47 INFO AbstractConnector: Stopped Spark@213e3629{HTTP/1.1,[http/1.1]}{0.0.0.0:4043}
20/12/10 21:29:47 INFO SparkUI: Stopped Spark web UI at http://team5:4043
20/12/10 21:29:47 INFO YarnClientSchedulerBackend: Interrupting monitor thread
20/12/10 21:29:47 INFO YarnClientSchedulerBackend: Shutting down all executors
20/12/10 21:29:47 INFO YarnSchedulerBackend$YarnDriverEndpoint: Asking each executor to shut down
20/12/10 21:29:47 INFO SchedulerExtensionServices: Stopping SchedulerExtensionServices
```

4. Text-to-Text introduction

Encoder and Decoder

The encoder and decoder blocks are actually multiple identical encoders and decoders stacked on top of each other. Both the encoder stack and the decoder stack have the same number of units. The encoder and the decoder stack works in the following way: The word embeddings of the input sequence are passed to the first encoder. These are then transformed and propagated to the next encoder. The output from the last encoder in the encoder-stack is passed to all the decoders in the decoder-stack as shown in the figure below [7]:

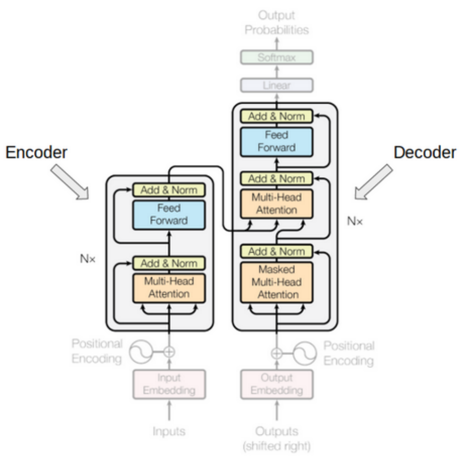


Figure 2. encoder and decoder structure

Attention
An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key [7].

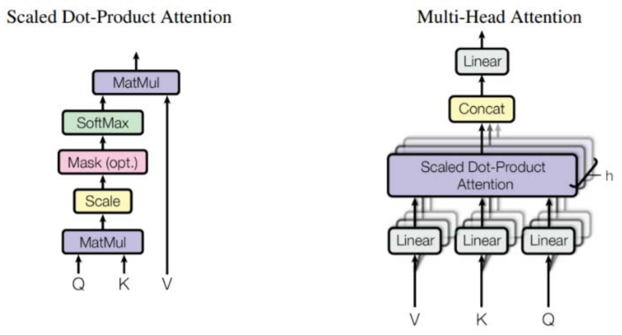
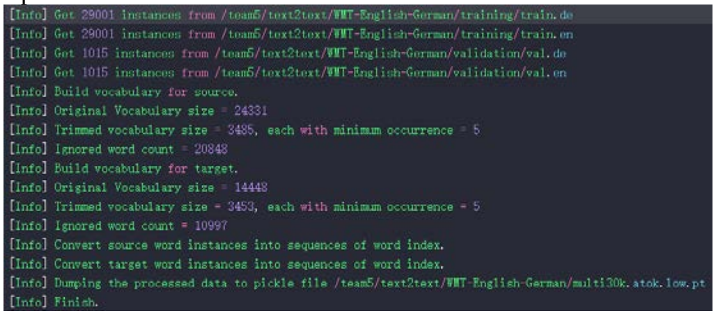


Figure 3: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel.

5. Text-to-Text configuration and testing procedures

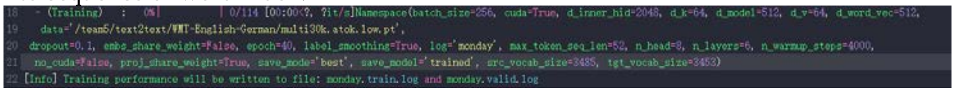
The explanation of the output is shown below.



```
[Info] Get 29001 instances from /team5/text2text/WMT-English-German/training/train.de
[Info] Get 29001 instances from /team5/text2text/WMT-English-German/training/train.en
[Info] Get 1015 instances from /team5/text2text/WMT-English-German/validation/val.de
[Info] Get 1015 instances from /team5/text2text/WMT-English-German/validation/val.en
[Info] Build vocabulary for source.
[Info] Original Vocabulary size = 24331
[Info] Trimmed vocabulary size = 3485, each with minimum occurrence = 5
[Info] Ignored word count = 20848
[Info] Build vocabulary for target.
[Info] Original Vocabulary size = 14448
[Info] Trimmed vocabulary size = 3453, each with minimum occurrence = 5
[Info] Ignored word count = 10997
[Info] Convert source word instances into sequences of word index.
[Info] Convert target word instances into sequences of word index.
[Info] Dumping the processed data to pickle file /team5/text2text/WMT-English-German/multi30k.atok.low.pt
[Info] Finish.
```

Figure 4. output of the preprocess

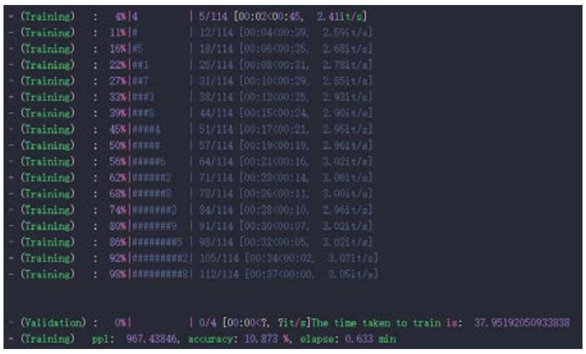
The figure 3 showed where the program got the training set and validation set. And it built the vocabulary and convert the word instance into sequence of word index.



```
  - (Training)   :   0%|          | 0/114 [00:00<?, ?it/s]Namespace(batch_size=256, cuda=True, d_inner_hid=2048, d_k=64, d_model=512, d_v=64, d_word_vec=512,
   data='/team5/text2text/WMT-English-German/multi30k.atok.low.pt',
 dropout=0.1, embs_share_weight=False, epoch=40, label_smoothing=True, log='monday', max_token_seq_len=52, n_head=8, n_layers=6, n_warmup_steps=4000,
  no_cuda=False, proj_share_weight=True, save_mode='best', save_model='trained', src_vocab_size=3485, tgt_vocab_size=3453)
[Info] Training performance will be written to file: monday.train.log and monday.valid.log
```

Figure 5. output of the parameter of training

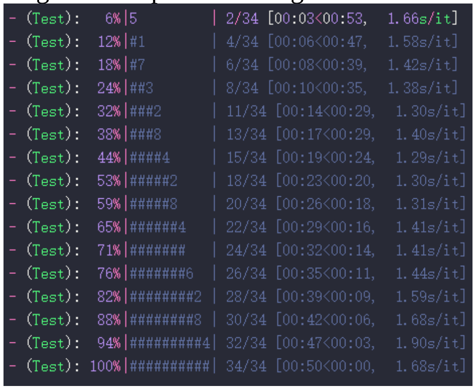
Figure 3 showed the setting of parameter of training. The word vector, the epoch, the layer of the encoder/decoder, the inner hide layer, whether use cuda, the rate of dropping out , whether label smoothing and the size of batch.



```
- (Training)   :   4%|4          | 5/114 [00:02<00:45,  2.41it/s]
- (Training)   :  11%|#          | 12/114 [00:04<00:39,  2.59it/s]
- (Training)   :  16%|#5         | 18/114 [00:06<00:35,  2.68it/s]
- (Training)   :  22%|##1        | 25/114 [00:08<00:31,  2.78it/s]
- (Training)   :  27%|##7        | 31/114 [00:10<00:29,  2.85it/s]
- (Training)   :  33%|###3       | 38/114 [00:12<00:25,  2.93it/s]
- (Training)   :  39%|###8       | 44/114 [00:15<00:24,  2.90it/s]
- (Training)   :  45%|####4      | 51/114 [00:17<00:21,  2.95it/s]
- (Training)   :  50%|#####      | 57/114 [00:19<00:19,  2.96it/s]
- (Training)   :  56%|#####6     | 64/114 [00:21<00:16,  3.02it/s]
- (Training)   :  62%|######2    | 71/114 [00:23<00:14,  3.06it/s]
- (Training)   :  68%|######8    | 78/114 [00:26<00:11,  3.00it/s]
- (Training)   :  74%|#######3   | 84/114 [00:28<00:10,  2.96it/s]
- (Training)   :  80%|#######9   | 91/114 [00:30<00:07,  3.02it/s]
- (Training)   :  86%|########5  | 98/114 [00:32<00:05,  3.02it/s]
- (Training)   :  92%|#########2| 105/114 [00:34<00:02,  3.07it/s]
- (Training)   :  98%|#########8| 112/114 [00:37<00:00,  3.05it/s]


- (Validation) :   0%|          | 0/4 [00:00<?, ?it/s]The time taken to train is:  37.95192050933838
- (Training)   ppl:  967.43846, accuracy: 10.873 %, elapse: 0.633 min
```

Figure 6. output of training and validation



```
- (Test):   6%|5          | 2/34 [00:03<00:53,  1.66s/it]
- (Test):  12%|#1         | 4/34 [00:06<00:47,  1.58s/it]
- (Test):  18%|#7         | 6/34 [00:08<00:39,  1.42s/it]
- (Test):  24%|##3        | 8/34 [00:10<00:35,  1.38s/it]
- (Test):  32%|###2       | 11/34 [00:14<00:29,  1.30s/it]
- (Test):  38%|###8       | 13/34 [00:17<00:29,  1.40s/it]
- (Test):  44%|####4      | 15/34 [00:19<00:24,  1.29s/it]
- (Test):  53%|#####2     | 18/34 [00:23<00:20,  1.30s/it]
- (Test):  59%|#####8     | 20/34 [00:26<00:18,  1.31s/it]
- (Test):  65%|######4    | 22/34 [00:29<00:16,  1.41s/it]
- (Test):  71%|#######    | 24/34 [00:32<00:14,  1.41s/it]
- (Test):  76%|#######6   | 26/34 [00:35<00:11,  1.44s/it]
- (Test):  82%|########2  | 28/34 [00:39<00:09,  1.59s/it]
- (Test):  88%|########8  | 30/34 [00:42<00:06,  1.68s/it]
- (Test):  94%|#########4| 32/34 [00:47<00:03,  1.90s/it]
- (Test): 100%|##########| 34/34 [00:50<00:00,  1.68s/it]
```

Figure 7. output of testing

Figure 6 showed the training time of training and validation for each epoch. The ppl means the training loss. The accuracy is the accuracy that predicted. The elapse is the total time for training or validation. We can see that there is some number like 6/114, the 114 means the total number of batch. There are 29,000 lines in the train.en and train.de. In this program, 256 is set for the batch size. 29,000/256 = 114. In the next section, the accuracy and training time will be analyzed.*C. Experimental Results and Analysis*

There are two sets of experiments. The first set of experiments modify the input data size. The second set of experiments modify the number of mappers and reducers. The corresponding performance metrices are calculated and analyzed.

6. Text-to-Text output and analysis of the program.

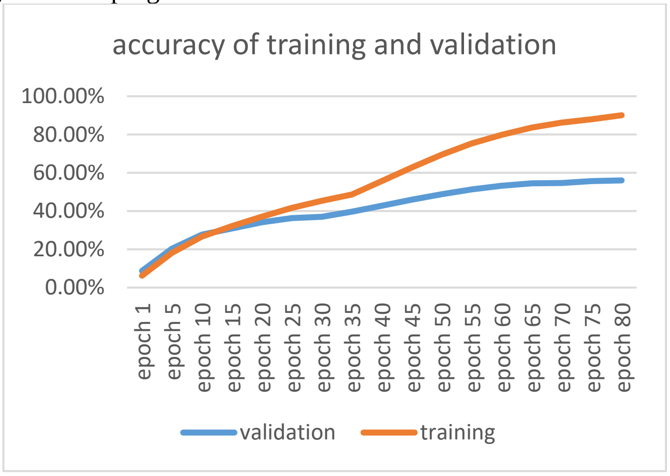
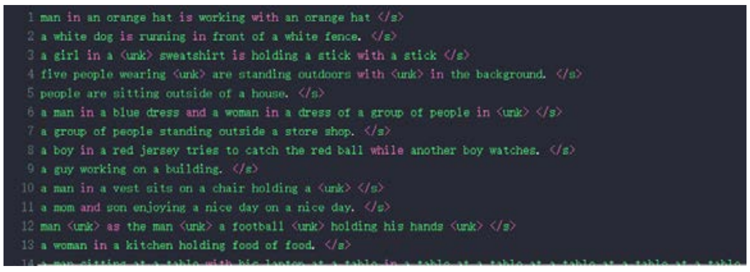


Figure 8. the accuracy of training and validation of Vietnamese to English with epoch 80 and batch size 128

The result is similar with English to Vietnamese with epoch 80 and batch size 128 in page 4.



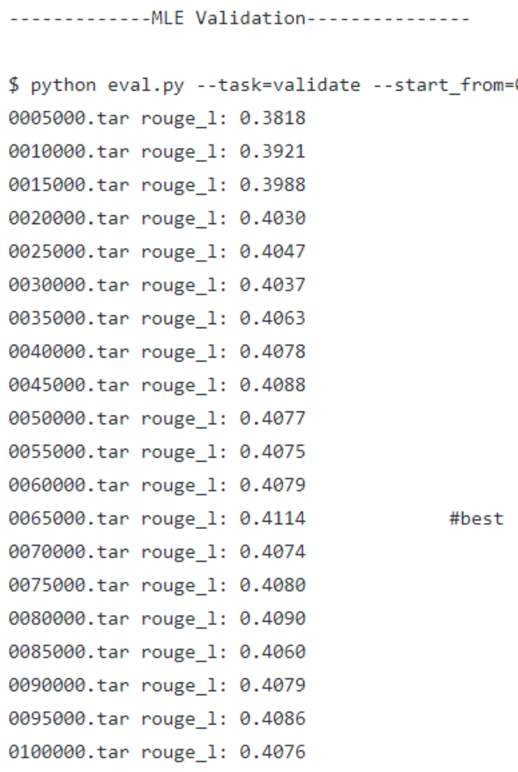
```
man in an orange hat is working with an orange hat </s>
a white dog is running in front of a white fence. </s>
a girl in a <unk> sweatshirt is holding a stick with a stick </s>
five people wearing <unk> are standing outdoors with <unk> in the background. </s>
people are sitting outside of a house. </s>
a man in a blue dress and a woman in a dress of a group of people in <unk> </s>
a group of people standing outside a store shop. </s>
a boy in a red jersey tries to catch the red ball while another boy watches. </s>
a guy working on a building. </s>
a man in a vest sits on a chair holding a <unk> </s>
a mom and son enjoying a nice day on a nice day. </s>
man <unk> as the man <unk> a football <unk> holding his hands <unk> </s>
a woman in a kitchen holding food of food. </s>
```

Figure 9. translated result of test.de

7. The formula to calculate theoretical GPU Flops:

$$FP32 = NVIDIA\ CUDA\ Cores \times GPU\ Boost\ Clock\ \times 2$$

8. Change the execution path for AIStation (Text Summarization):



```
-------------MLE Validation---------------

$ python eval.py --task=validate --start_from=0005000.tar
0005000.tar rouge_l: 0.3818
0010000.tar rouge_l: 0.3921
0015000.tar rouge_l: 0.3988
0020000.tar rouge_l: 0.4030
0025000.tar rouge_l: 0.4047
0030000.tar rouge_l: 0.4037
0035000.tar rouge_l: 0.4063
0040000.tar rouge_l: 0.4078
0045000.tar rouge_l: 0.4088
0050000.tar rouge_l: 0.4077
0055000.tar rouge_l: 0.4075
0060000.tar rouge_l: 0.4079
0065000.tar rouge_l: 0.4114            #best
0070000.tar rouge_l: 0.4074
0075000.tar rouge_l: 0.4080
0080000.tar rouge_l: 0.4090
0085000.tar rouge_l: 0.4060
0090000.tar rouge_l: 0.4079
0095000.tar rouge_l: 0.4086
0100000.tar rouge_l: 0.4076
```

9. Shell instruction used for text summarization:
MLE:
```
python -u /team5/DC_AIBench_Component/PyTorch/Text_summarization/train.py
```
MLE+RL:
```
python -u /team5/DC_AIBench_Component/PyTorch/Text_summarization/train.py --train_mle=yes --train_rl=yes --mle_weight=0.25 --load_model=0015000.tar --new_lr=0.0001
```
RL:
```
python -u  /team5/DC_AIBench_Component/PyTorch/Text_summarization/eval.py --task=test --load_model=0020000.tar
```
Evaluation:
```
python -u  /team5/DC_AIBench_Component/PyTorch/Text_summarization/eval.py --task=test --load_model=0020000.tar
```
10. Example of the Evaluation Log:
```
 Finished constructing vocabulary of 25000 total words. Last word added: stowaway
0020000.tar (MLE)
scores: {'rouge-1': {'f': 0.43681373574350985, 'p': 0.4779883475783452, 'r': 0.4187843334711541}, 'rouge-2': {'f': 0.23082822009187096, 'p': 0.25367488899988866, 'r': 0.2215794210387702}, 'rouge-l': {'f': 0.41950314892375046, 'p': 0.46035252775002555, 'r': 0.40147296693416734}}
0020000.tar  (MLE+RL)
scores: {'rouge-1': {'f': 0.4391358166088737, 'p': 0.4799450433172477, 'r': 0.4213217018781728}, 'rouge-2': {'f': 0.23292165216376548, 'p': 0.2553191155141156, 'r': 0.2239357220319525}, 'rouge-l': {'f': 0.4225161082250343, 'p': 0.4633210619935595, 'r': 0.40452482727139794}}
0020000.tar (RL)
scores: {'rouge-1': {'f': 0.32340831368721407, 'p': 0.26406063646773426, 'r': 0.44457277608002793}, 'rouge-2': {'f': 0.15693398485449994, 'p': 0.12857896733011795, 'r': 0.2194150879738877}, 'rouge-l': {'f': 0.43909209931753823, 'p': 0.4773794745994712, 'r': 0.43050079922800344}}
```
11. Download the training and evaluation data for Gigaword used in text summarization:
https://drive.google.com/open?id=0B6N7tANPyVeBNmlSX19Ld2xDU1E
12. An example of the training data used for text summarization:

abstract.6.4.2american guard sinks free throws for finnish title.

article american guard lonnie cooper sank five crucial free throws in the final ## seconds sunday as honka of espoo beat <unk> lahti ##-## to win its second straight finnish basketball title

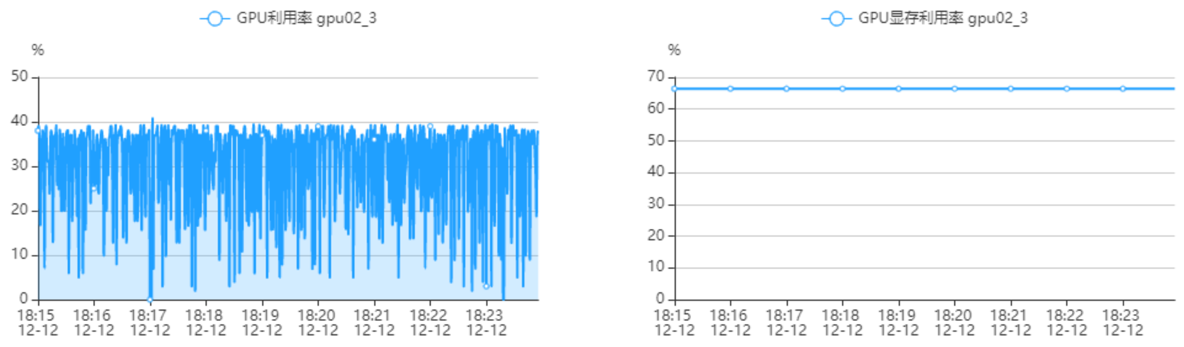
13. How to change the vocabulary size and the batch size:
    This file is under the **data_util** directory. In the **config.py** you can change the hyper parameters of the training there.
14. Training log given by the developer [20]:

We can learn from the log that 65000 training iteration performs best during the validation.

```
-------------MLE Validation---------------

$ python eval.py --task=validate --start_from=0005000.tar
0005000.tar rouge_l: 0.3818
0010000.tar rouge_l: 0.3921
0015000.tar rouge_l: 0.3988
0020000.tar rouge_l: 0.4030
0025000.tar rouge_l: 0.4047
0030000.tar rouge_l: 0.4037
0035000.tar rouge_l: 0.4063
0040000.tar rouge_l: 0.4078
0045000.tar rouge_l: 0.4088
0050000.tar rouge_l: 0.4077
0055000.tar rouge_l: 0.4075
0060000.tar rouge_l: 0.4079
0065000.tar rouge_l: 0.4114            #best
0070000.tar rouge_l: 0.4074
0075000.tar rouge_l: 0.4080
0080000.tar rouge_l: 0.4090
0085000.tar rouge_l: 0.4060
0090000.tar rouge_l: 0.4079
0095000.tar rouge_l: 0.4086
0100000.tar rouge_l: 0.4076
```

15. An Example of the GPU Resource Usage for Text Summarization



16. Lossextract.py

```python
import xlwt
import re
start = 1000
max_iter = 20000
book=xlwt.Workbook(encoding="utf-8",style_compression=0)
sheet = book.add_sheet('sheet01', cell_overwrite_ok=True)
sheet.write(0,0,'Iter')
sheet.write(0,1,'MLE_Loss')
sheet.write(0,2,'Reward')
with open('data3') as f:
    count = 1
    for line in f:
        match = re.match(r"iter: (.+?) mle_loss: (.+?) reward: (.+?)", line)
        sheet.write(count, 0, float(match.group(1)))
        sheet.write(count, 1, float(match.group(2)))
        sheet.write(count, 2, float(match.group(3)))
        count += 1
book.save('test3.xls')
```

17. Histogram.py

```
import matplotlib.pyplot as plt
import numpy as np
length = []
with open('train.article.txt', encoding='utf-8')as f:
    for line in f:
        length.append(len(line))
print(len(length))
plt.hist(x=length, bins=30, weights=np.ones(len(length)) / len(length), histtype='stepfilled', 
color='steelblue', edgecolor='black')
plt.xlabel("Length")
plt.ylabel("Percentage")
plt.title("Analysis of Article Length")
plt.show()
```

18. Result of the Summarization:

MLE

article: dorothy porter wesley , a librarian who played the primary role in building howard university 's major collection of books and other materials for the study of black history and culture , died sunday in fort lauderdale , fla. , at the home of her daughter , where she had moved recently from washington .
ref: dorothy porter wesley howard university librarian at ##
dec: dorothy porter wesley ## librarian

article: big hitting lindsay davenport , who missed the pre-wimbledon grass court season to attend her graduation ceremony at murrieta valley high school in california , reached the last ## of the women 's singles at wimbledon on friday when she scored a three-set win over germany 's barbara rittner .
ref: davenport powers into the <unk>
dec: davenport reaches last ## at wimbledon

article: nancy white , the white-gloved editor of harper 's bazaar during the ####s , when it captured the spirit of the decade with evocative photographs of models in bikinis , boots and big watches , not to mention spacesuits , died on saturday at her home in manhattan .
ref: nancy white ## edited harper 's bazaar in ####s
dec: nancy white editor of harper 's bazaar dies at ##

MLE+RL

article: dorothy porter wesley , a librarian who played the primary role in building howard university 's major collection of books and other materials for the study of black history and culture , died sunday in fort lauderdale , fla. , at the home of her daughter , where she had moved recently from washington .
ref: dorothy porter wesley howard university librarian at ##
dec: dorothy wesley wesley ## dies

article: big hitting lindsay davenport , who missed the pre-wimbledon grass court season to attend her graduation ceremony at murrieta valley high school in california , reached the last ## of the women 's singles at wimbledon on friday when she scored a three-set win over germany 's barbara rittner .
ref: davenport powers into the <unk>
dec: davenport reaches last ##

article: nancy white , the white-gloved editor of harper 's bazaar during the ####s , when it captured the spirit of the decade with evocative photographs of models in bikinis , boots and big watches , not to mention spacesuits , died on saturday at her home in manhattan .
ref: nancy white ## edited harper 's bazaar in ####s
dec: nancy white ## editor of harper 's bazaar

RL

article: dorothy porter wesley , a librarian who played the primary role in building howard university 's major collection of books and other materials for the study of black history and culture , died sunday in fort lauderdale , fla. , at the home of her daughter , where she had moved recently from washington .
ref: dorothy porter wesley howard university librarian at ##
dec: dorothy porter wesley ## librarian dorothy wesley dies at ## ## ## ## ## ##

article: big hitting lindsay davenport , who missed the pre-wimbledon grass court season to attend her graduation ceremony at murrieta valley high school in california , reached the last ## of the women 's singles at wimbledon on friday when she scored a three-set win over germany 's barbara rittner .
ref: davenport powers into the <unk>
dec: davenport reaches wimbledon ## in wimbledon ##

article: nancy white , the white-gloved editor of harper 's bazaar during the ####s , when it captured the spirit of the decade with evocative photographs of models in bikinis , boots and big watches , not to mention spacesuits , died on saturday at her home in manhattan .
ref: nancy white ## edited harper 's bazaar in ####s
dec: nancy white ## editor editor nancy white dies at ## ## ## ## ## ##

VII. Reference

[1] L. Breiman. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324
[2] W. Koehrsen. Random Forest Simple Explanation. Retrieved from: https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d. Accessed on 2020/12/13.
[3] J. H. Friedman. "Greedy function approximation: A gradient boosting machine," in Annals of Statistics 29 (2001), pp. 1189–1232.
[4] J. Ye, J. Chow, J. Chen, and Z. Zhao. "Stochastic gradient boosted distributed decision trees," in Proceedings of the 18th ACM Conference on Information and Knowledge Management (2009), pp. 2061-2064.
[5] Intel, Intel-bigdata/HiBench, Github. Accessed on: December 9, 2020. [Online]. Available: https://github.com/Intel-bigdata/HiBench
[6] K. Hwang, Cloud Computing for Machine Learning and Cognitive Applications, vol. 8, Massachusetts: MIT, 2016.
[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, ... & I, Polosukhin. (2017). Attention is all you need. Advances In Neural Information Processing Systems, 30, 5998-6008.
[8] M. T. Luong, & C. D. Manning. (2015, December). Stanford neural machine translation systems for spoken language domains. In Proceedings of the International Workshop on Spoken Language Translation (pp. 76-79).
[9] Stanford. Neural Machine Translation. Retrieved from https://nlp.stanford.edu/projects/nmt/
[10] AIBench: *AIBench: A Datacenter AI Benchmark Suite, BenchCouncil – Summary*, https://www.benchcouncil.org/AIBench/index.html
[11] AiRS (Shenzhen Institute of Artificial Intelligence And Robotics for Society): *Introduction of Intelligence Cloud Systen, 5G and IoT Edge Computing Platform (智能云系统、5G 与 IOT 边缘计算平台简介)*
[12] Nvidia Official: *GEFORCE RTX 2080 Ti Specs* https://www.nvidia.com/en-us/geforce/graphics-cards/rtx-2080-ti/
[13] Inspur: *Product Overview of AIStation*: https://en.inspur.com/en/2402530/2402532/2402814/2402822/2406367/index.html
[14] Abigail See, Peter J.Liu, Christopher D. Manning: *Get To The Point: Summarization with Pointer-Generator Network,* https://arxiv.org/pdf/1704.04368.pdf
[15] Ramesh Nallapati, Bowen Zhou, Cicero dos Santos, Gulcehrc, Bin Xia: *Abstractive Text Summarization using Sequence-to-sequence RNNs and Beyond,* https://arxiv.org/pdf/1602.06023.pdf
[16] Romain Paulus, Caiming Xiong, Richard Socher: *A DEEP REINFORCED MODEL FOR ABSTRACTIVE SUMMARIZATION,* https://arxiv.org/pdf/1705.04304.pdf
[17] Rohith Reddy: *Text-Summarizer-Pytorch*, https://github.com/rohithreddy024/Text-Summarizer-Pytorch
[18] Jason Brownlee: *Difference Between a Batch and an Epoch in a Nueral Network,* https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/
[19] https://medium.com/towards-artificial-intelligence/understanding-abstractive-text-summarization-from-scratch-baaf83d446b3
[20] Chin-Yew Lin: *ROUGE: A Package for Automatic Evaluation of Summaries* http://citeseer.ist.psu.edu/viewdoc/download;jsessionid=368956B77243860A93FC0E3726EE7919?doi=10.1.1.111.9426&rep=rep1&type=pdf