**Online Payment Fraud Detection**

Team 11 - Zihan Wan & Feifan Tai

卡通人物

中度可信度描述已自动生成

**1. Introduction**

**1.1 Background**

With the development and popularity of online and mobile payments, people still face a considerable risk of payment fraud while enjoying this convenience. Under the cover of malicious technology, fraudsters are becoming more sophisticated, and the amount of fraud continues to rise. Therefore, it is crucial to identify suspicious transactions quickly and accurately.

**1.2 Motivation**

For enterprises, payment fraud not only hurts revenue, but also undermines consumer confidence and even corporate reputation. So, they need to proactively assess their exposure to fraud risk and respond appropriately. For individual consumers, payment fraud can have more serious consequences, even affecting lives.

In the face of the increasing volume of payment transaction data and the rapidly changing market, machine learning algorithm models need to be constantly iterated and updated, and parallel computing methods can undoubtedly save a lot of time and labor costs.

**1.3 Goal**

We plan to use machine learning to identify online payment fraud so that we can better understand the prevalent characteristics of fraudulent transactions. We also intend to perform data processing and model training on multiple CPUs and multiple GPUs with the help of parallel computing methods to achieve speedup and improve accuracy, and finally analyze and summarize the different results.

**2. Data Description**

Source Link: <https://www.kaggle.com/datasets/jainilcoder/online-payment-fraud-detection>

图形用户界面, 应用程序

描述已自动生成

This dataset (493.53 MB) contains historical information about fraudulent transactions, its basic information is as follows:

|  |  |  |
| --- | --- | --- |
| **Data Size** | 6362620 rows × 11 columns | |
| **Data Types** | int64(×3) / float64(×5) / object (×3) | |
| **Target** | “isFraud” (1/0) | |
| **Features** | step | time units, 1 step = 1 hour |
| type | type of online transaction (payment / transfer / ...) |
| amount | amount of the transaction |
| nameOrig | name of the customer starting the transaction |
| oldbalanceOrg | balance before the transaction |
| newbalanceOrig | balance after the transaction |
| nameDest | name of the customer receiving the transaction |
| oldbalanceDest | initial balance of recipient before the transaction |
| newbalanceDest | new balance of recipient after the transaction |
| isFraud | whether it is a fraudulent transaction? |
| isFlaggedFraud | Is the transaction identified as fraudulent? |

**3. Methodology**

* Write a function to create metadata for the initial data.
* Executing the function serially.
* Execute the function in parallel using Multiprocessing Pool and Multiprocessing Process, respectively.
* Data processing and feature engineering using NumPy Array and Pandas.
* Data processing and feature engineering using Dask Array and Dask DataFrame.
* Handling imbalance data by resampling technique using Pandas and Dask.
* Training the model with XGBoost.
* Adjusting the model parameters by Grid Search with Multiple-GPU and Multiple-CPU.
* Using K-fold cross-validation to evaluate the model with Multiple-GPU and Multiple-CPU.

**4. Results and Analysis**

**4.1 Environment Description**

**4.1.1 CLUSTER**

Cluster: Discovery High-Performance Computing Cluster

Reservation: csye7105-gpu

Reservation memory: 100 GB

**4.1.2 GPU:**

Model: Nvidia V100-SXM2

GPU count: 1,2 and 4

GPU Memory: 32.4805 GB / GPU

Clock: 1.290Ghz with max boost of 1.530GHz

**4.1.3 CPU:**

Model: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz

CPU max MHz: 3500.0000

CPU min MHz: 1200.0000

Thread(s) per core: 2

CPUs: 48

**4.2 Code Files Description**

There are four code files, which are arranged according to the order of execution as follows.

**4.2.1 BeforeModel\_OOD.ipynb**

This jupyter file contains four main sections:

* Section1 - Reading data
* Section2 - Creating metadata
* Section3 - Feature engineering
* Section4 - Handling unbalanced data

For section1, after briefly viewing and filtering the data, we used a heat map to show the correlation between features(Figure 2). We found a strong positive correlation between balance before transaction and balance after transaction, which means that we need to do some feature engineering to solve it.

To get a more intuitive view of the data and improve the efficiency of data processing, for section2, we wrote a function to create metadata, taking the dataset itself and each feature as parameters, returning the name, variable type, data type, number of unique values, number of missing values, percentage of missing values and imputation method for each feature. The return result is as shown in the Figure 1.

表格

描述已自动生成

Figure 1

As shown in Figure xxx, the dataset is already pre-cleaned, so we don't need to spend much effort on data processing. We have tried serial method, Multiprocessing Pool method, and Multiprocessing Process method in total three ways while executing this function of creating metadata.

For section3, we used NumPy and Pandas, as well as Dask Array and Dask DataFrame, respectively, for the following operations.

First, for the nominal variable 'type', we encoded it using OneHotEncoder. Then, to deal with the problem mentioned in section1, we calculated the balance difference before and after the transaction and replaced the original features as new features. At this point, the heat map of the correlation between features is shown in Figure 3.

图表

描述已自动生成图表, 树状图

描述已自动生成

Figure 2 Figure 3

Finally, for feature selection, we plotted the feature importance graph(Figure 4) based on weight and gain, and selected five features based on comprehensive consideration.

表格

描述已自动生成

Figure 4

For section4, as for section3, we performed the following operations using Pandas and Dask, respectively.

Given our observation that our data are extremely unbalanced (Figure 5), we employ resampling techniques (both oversampling and undersampling) to process our data.

图表, 饼图

描述已自动生成

Figure 5

The distribution of the data after oversampling and undersampling are shown in Figure 6 and Figure 7, respectively.

图表, 饼图

描述已自动生成图表, 饼图

描述已自动生成

Figure 6 Figure 7

So far, we get the balanced data. For the Dask part, we divide the data into X and y and calculate the chunk size respectively. The results of the balanced data after oversampling are shown in Figure 8 and Figure 9.

表格

中度可信度描述已自动生成 图形用户界面

低可信度描述已自动生成

Figure 8 Figure 9

The results of the balanced data after undersampling are shown in Figure xxx and Figure xxx.

图片包含 表格

描述已自动生成图形用户界面

中度可信度描述已自动生成

Figure 10 Figure 11

Since the size of data after using undersampling is too small and its chunk size is also very small, we decided to use only the balanced data obtained after oversampling as the final data.

**4.2.2 ModelWithPandasOversampling.py**

This python file contains six main sections:

* Section1 - Reading data
* Section2 - Using pandas oversampleing the data
* Section3 - Building Model
* Section4 - Adjusting the model parameters by Grid Search
* Section5 - Using K-fold cross-validation to evaluate the model
* Section6 - Creating loop method runs on Cluster

In section1 and 2,we use Pandas to obtain the data and balance it.

In section3, we build model with XGBoost with 7 parameters which are max\_depth, learning\_rate, n\_estimators, min\_child\_weight, max\_delta\_step, subsample, reg\_alpha.

In section 4, we create a method to adjust the 7 parameters with Grid Search.

In section 5, we use K-fold cross-validation to evaluate the model

In section 6, we create loop method runs on cluster.

**4.2.3 ModelWithDaskOversampling.py**

This python file contains six main sections:

* Section1 - Reading data
* Section2 - Using pandas oversampleing the data
* Section3 - Building Model
* Section4 - Adjusting the model parameters by Grid Search
* Section5 - Using K-fold cross-validation to evaluate the model
* Section6 - Creating loop method runs on Cluster

In section1 and 2,we use Dask to obtain the data and balance it adjust chunksize to a appropriate size.

In section3, we build model with XGBoost with 7 parameters which are max\_depth, learning\_rate, n\_estimators, min\_child\_weight, max\_delta\_step, subsample, reg\_alpha.

In section 4, we create a method to adjust the 7 parameters with Grid Search.

In section 5, we use K-fold cross-validation to evaluate the model

In section 6, we create loop method runs on cluster.

**4.2.4 ResultsEvaluation\_OOD.ipynb**

This jupyter file contains two main sections, and each section is divided into two parts, Pandas and Dask

* Section1 – BeforeModel
* Section2 - Results Evaluation

In section1, we use Pandas and Dask to obtain the final data after balancing respectively.

In section2, we re-model the predictions using the best parameters modeled with different number of CPUs and different number of GPUs, and output the evaluation metrics (accuracy, precision, recall and f1 score). Some of the results are shown in Figure 12-15.

文本

描述已自动生成 文本

描述已自动生成

Figure 12 Figure 13

文本

描述已自动生成 文本

描述已自动生成

Figure 14 Figure 15

Finally, we also plot the elapsed time, speedup, and efficiency graphs to train the model in parallel with different number of CPUs and different number of GPUs.

**4.3 Parallel Results Analysis**

**4.3.1 Parallel execution to create metadata**

The elapsed time for each of the three methods are summarized in the following table:

|  |  |  |
| --- | --- | --- |
| **Execution Method** | **Elapsed Time(seconds)** | |
| Serial | 2.413 | |
| Pool | CPU=1 | 16.988 |
| CPU=2 | 15.885 |
| CPU=4 | 15.886 |
| CPU=8 | 16.419 |
| Process | Processes = 2 | 0.320 |

We first tried to use the Pool method, but it did not work as well as we expected, regardless of the number of CPUs. The execution time using the pool method is about 6 times longer than the serial execution time.

We analyzed that the causes might be that:

* This metadata creation function does not require a lot of CPU resources to parallelize.
* The slow startup time of additional processes and communication overhead.

To avoid the above, then we tried using Process method and manually splitting the workload list. Good results were obtained when we split work into n processes, and each process is responsible for 1/n columns.

When n=2, i.e. when two processes are running simultaneously, the elapsed time was more than 10 times faster than serial execution.

**4.3.2 Parallelizing Computations with Dask Arrays**

When we switch from NumPy Array to Dask Array, the feature construction(simple calculations) is more than 75 times faster, and the comparison results are shown in Figure 16 and Figure 17.

图形用户界面, 文本

描述已自动生成

Figure 16

文本

描述已自动生成

Figure 17

This can be attributed to Dask's powerful parallel technology that can handle large scale data and significantly improve computation time.

**4.3.3 Results of parallelizing the model training on Multiple-GPU and Multiple-CPU**

**Experiment 1: GPU = 1**

**GPU:** v100-sxm2

**Reservation:** csye7105-gpu

**Method:** Pandas, MultiProcessing

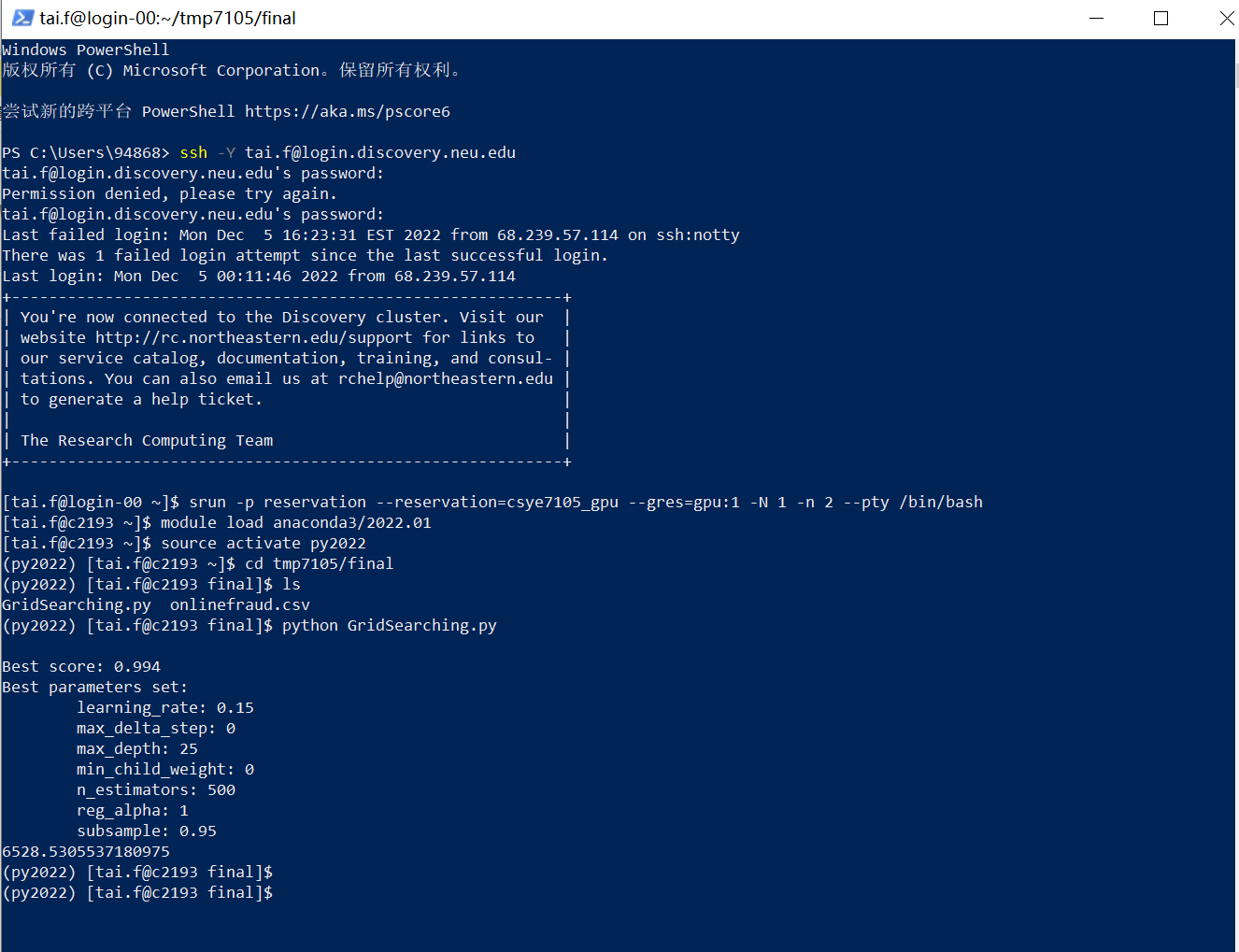


Figure 18

**Experiment 2: GPU = 2**

**GPU:** v100-sxm2

**Reservation:** csye7105-gpu

**Method:** Pandas, MultiProcessing

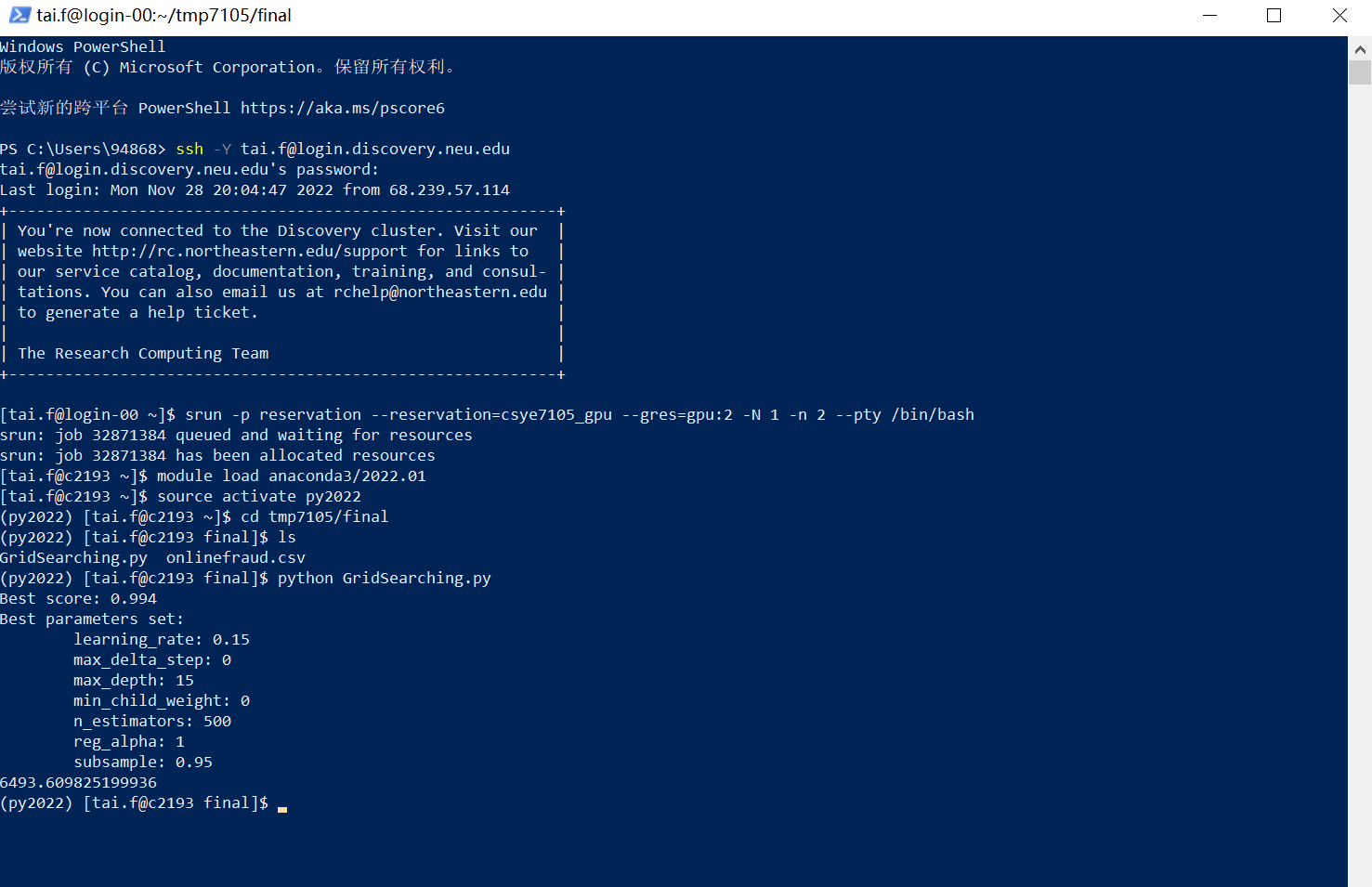


Figure 19

**Experiment 3: GPU = 4**

**GPU:** v100-sxm2

**Reservation:** csye7105-gpu

**Method:** Pandas, MultiProcessing

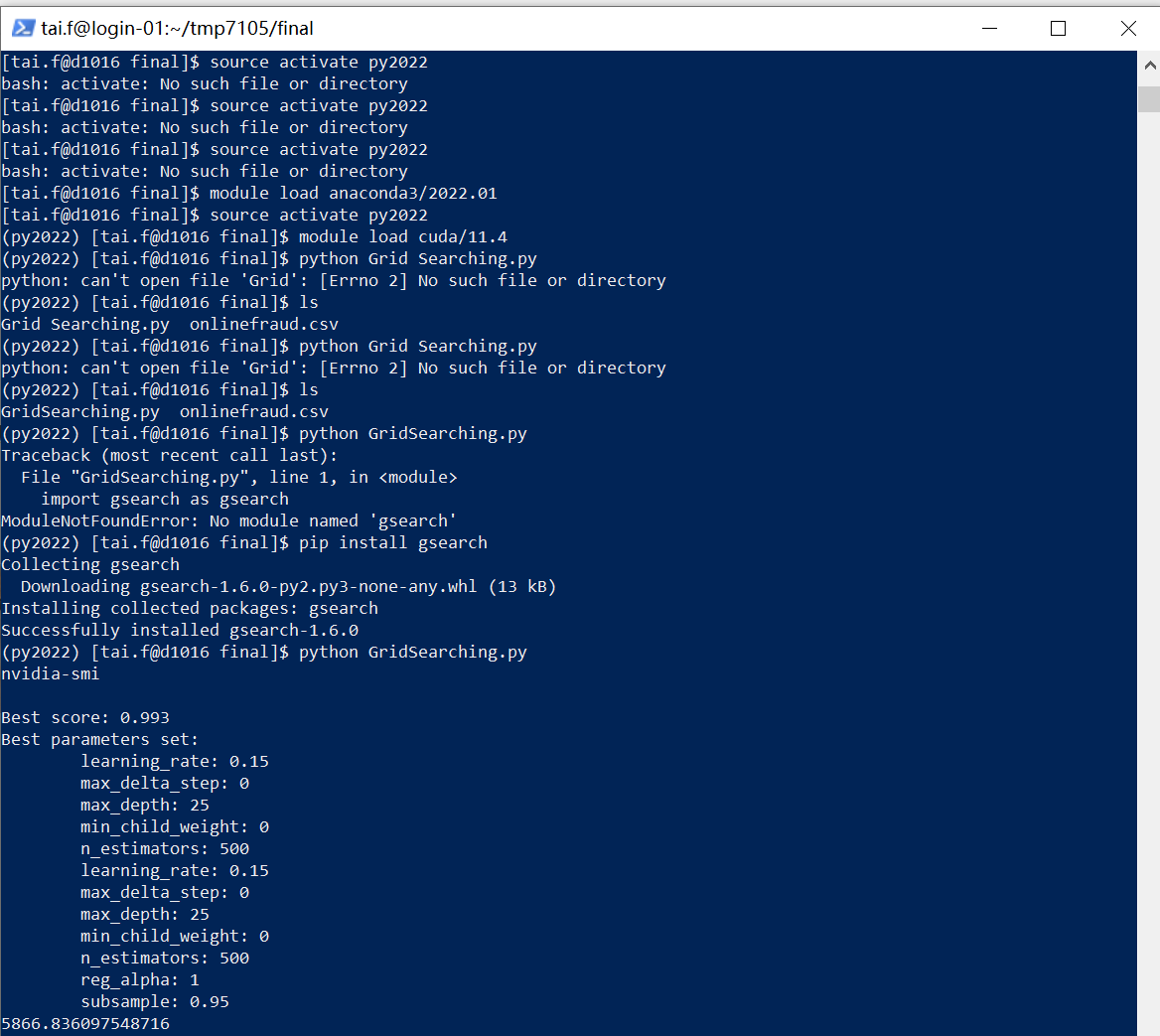


Figure 20

**Experiment 4: GPU = 1**

**GPU:** v100-sxm2

**Reservation:** csye7105-gpu

**Method:** Dask, MultiProcessing

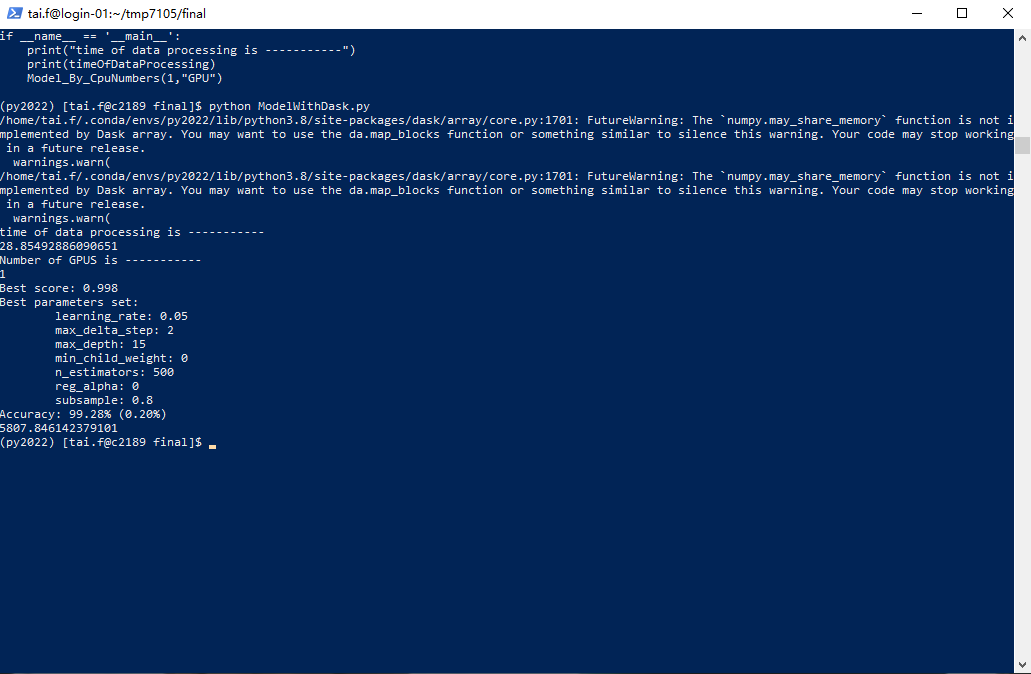


Figure 21

**Experiment 5: GPU = 2**

**GPU:** v100-sxm2

**Reservation:** csye7105-gpu

**Method:** Dask, MultiProcessing

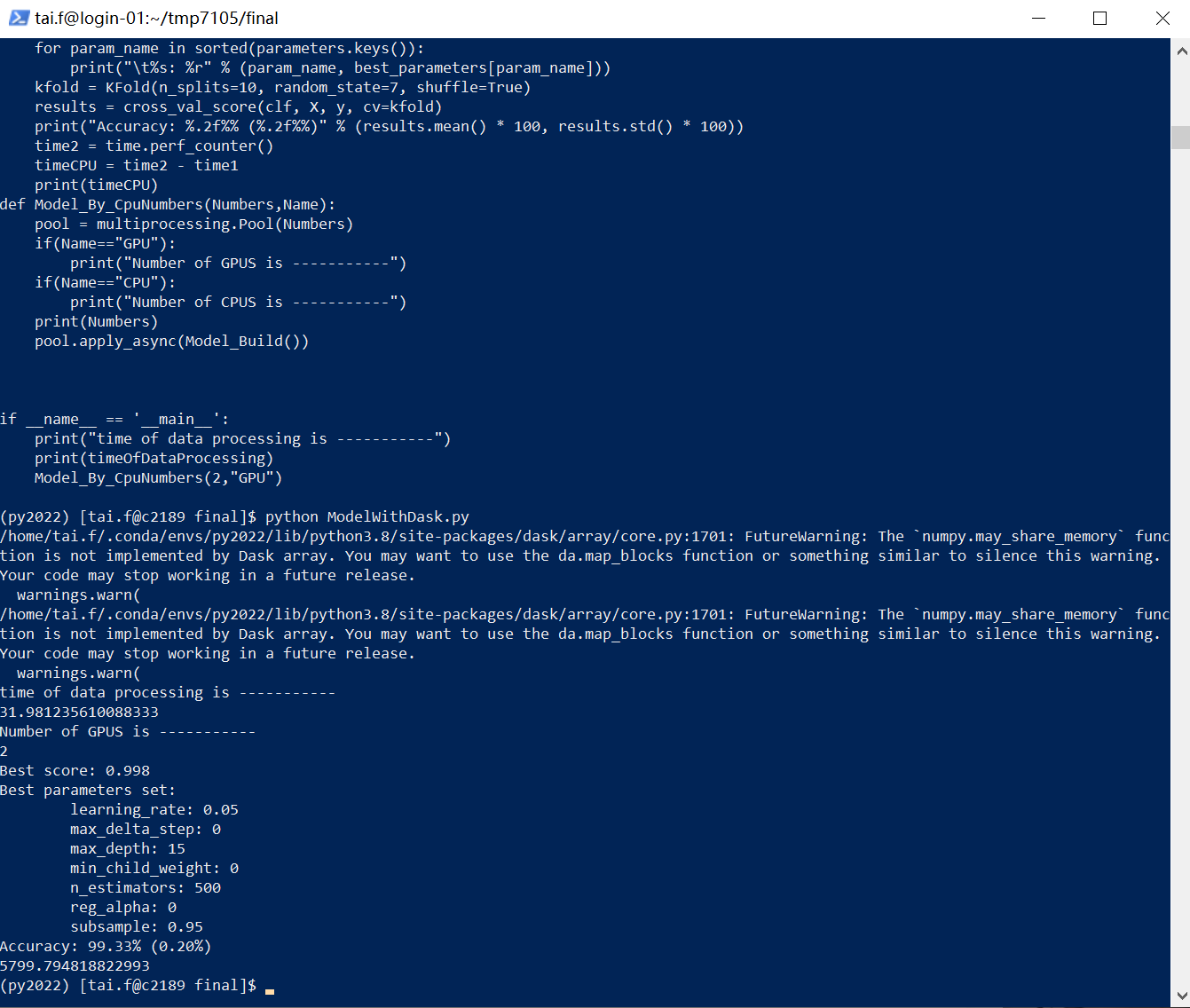


Figure 22

**Experiment 6: GPU = 4**

**GPU:** v100-sxm2

**Reservation:** csye7105-gpu

**Method:** Dask, MultiProcessing

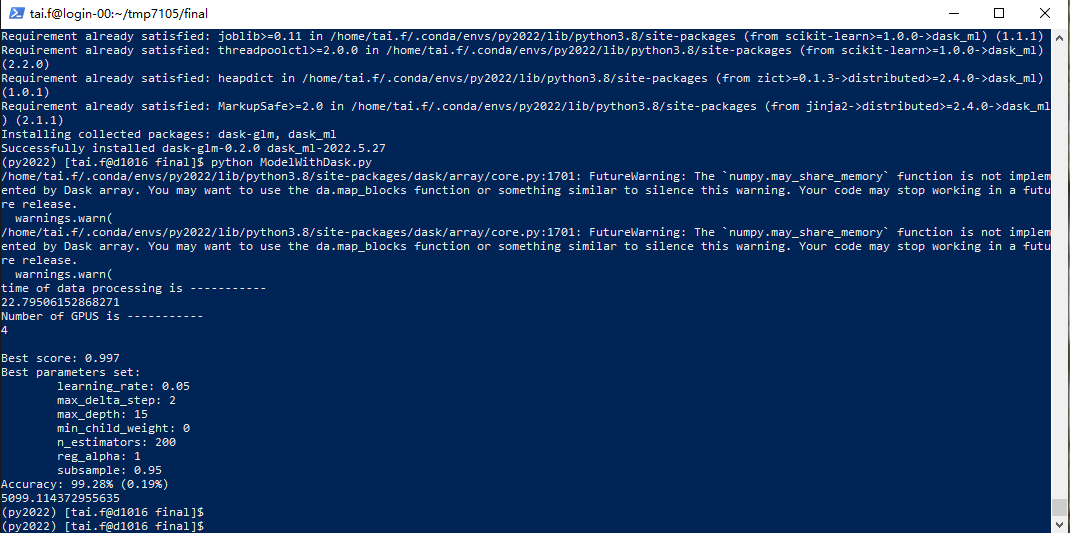


Figure 23

**Experiment 7: CPU = 1**

**GPU: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz**

**Reservation:** csye7105

**Method:** Pandas, MultiProcessing

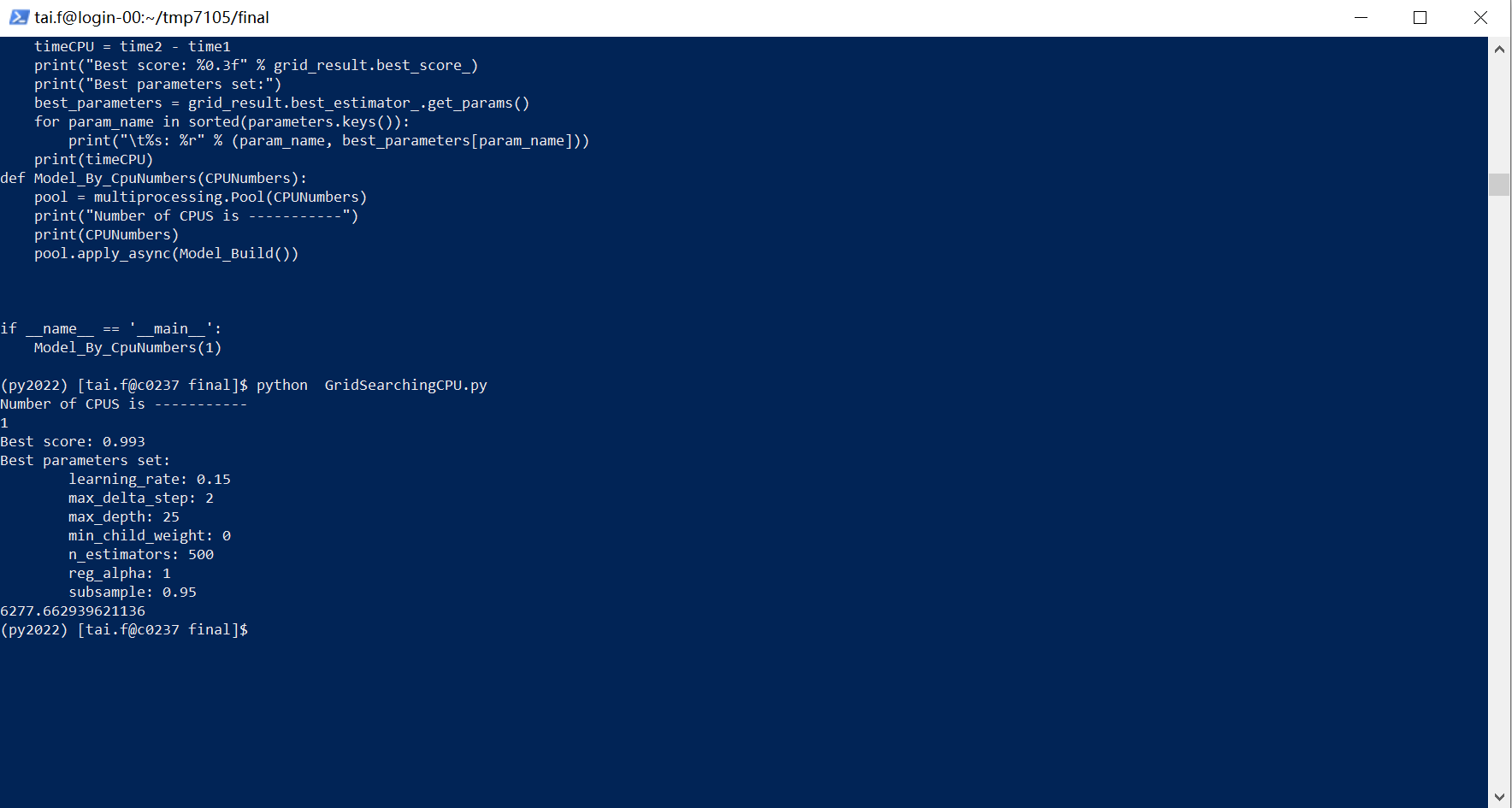


Figure 24

**Experiment 8: CPU = 2**

**GPU: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz**

**Reservation:** csye7105

**Method:** Pandas, MultiProcessing

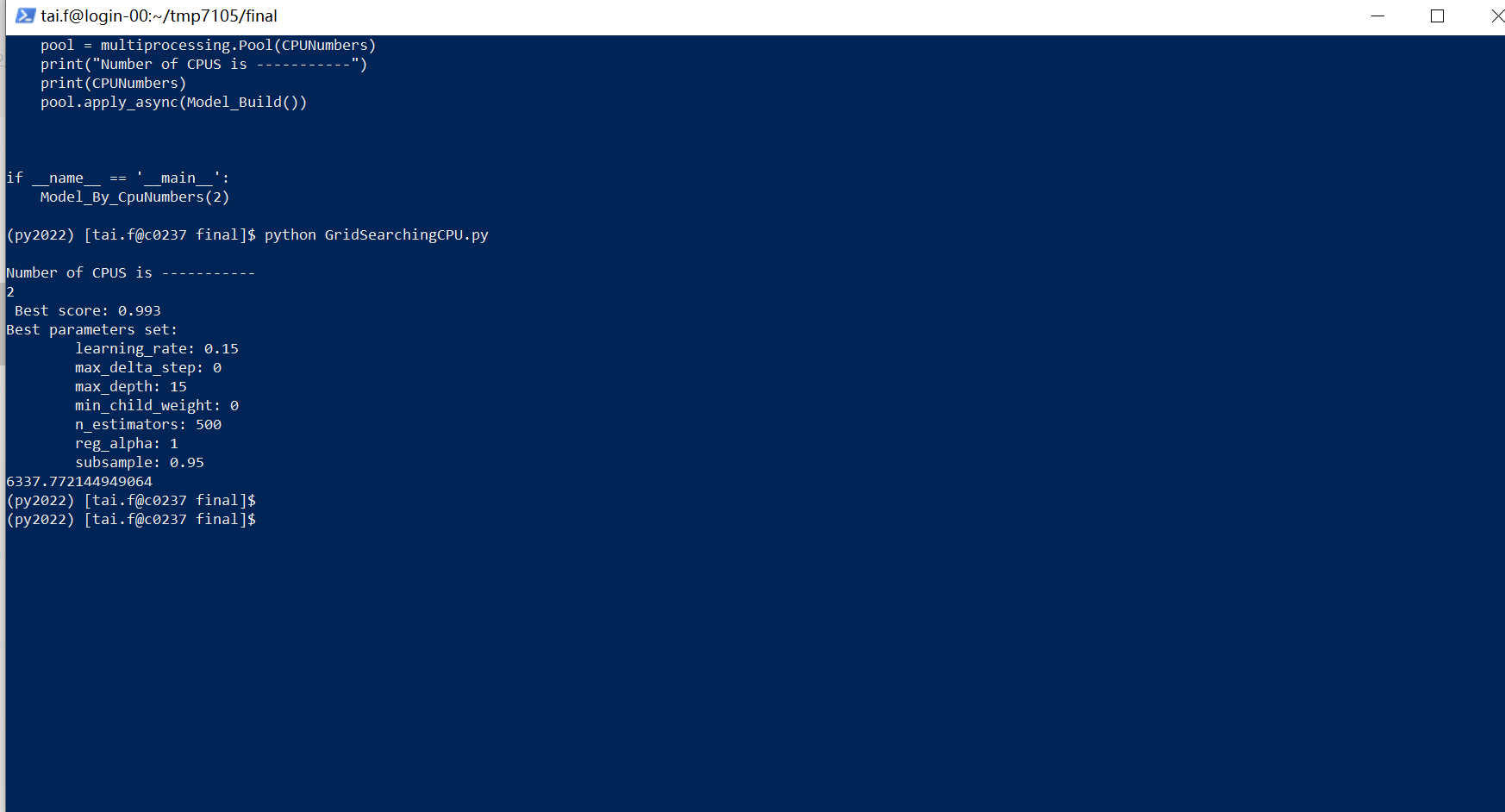


Figure 25

**Experiment 9: CPU = 4**

**GPU: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz**

**Reservation:** csye7105

**Method:** Pandas, MultiProcessing

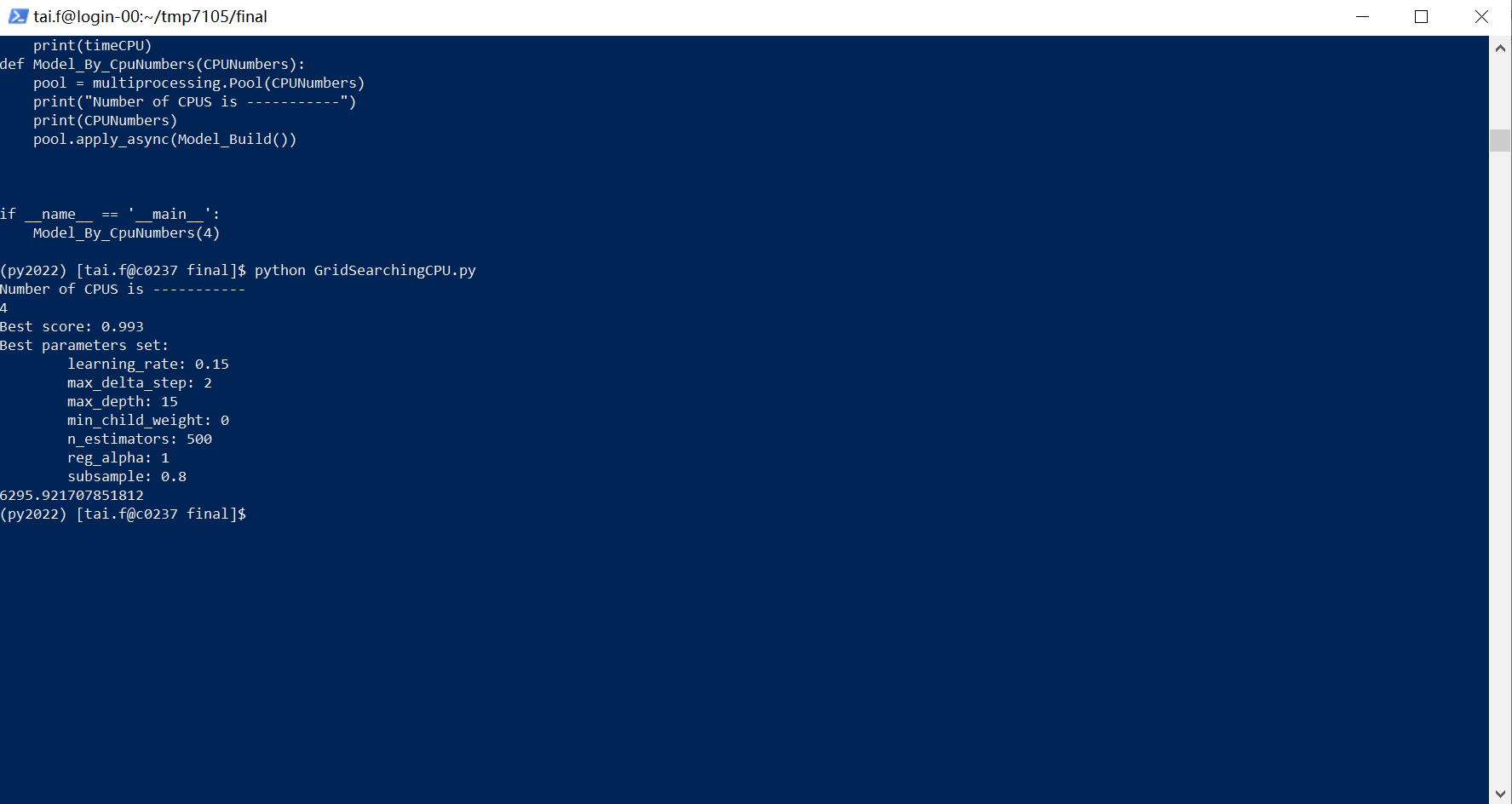


Figure 26

**Experiment 10: CPU = 8**

**GPU: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz**

**Reservation:** csye7105

**Method:** Pandas, MultiProcessing

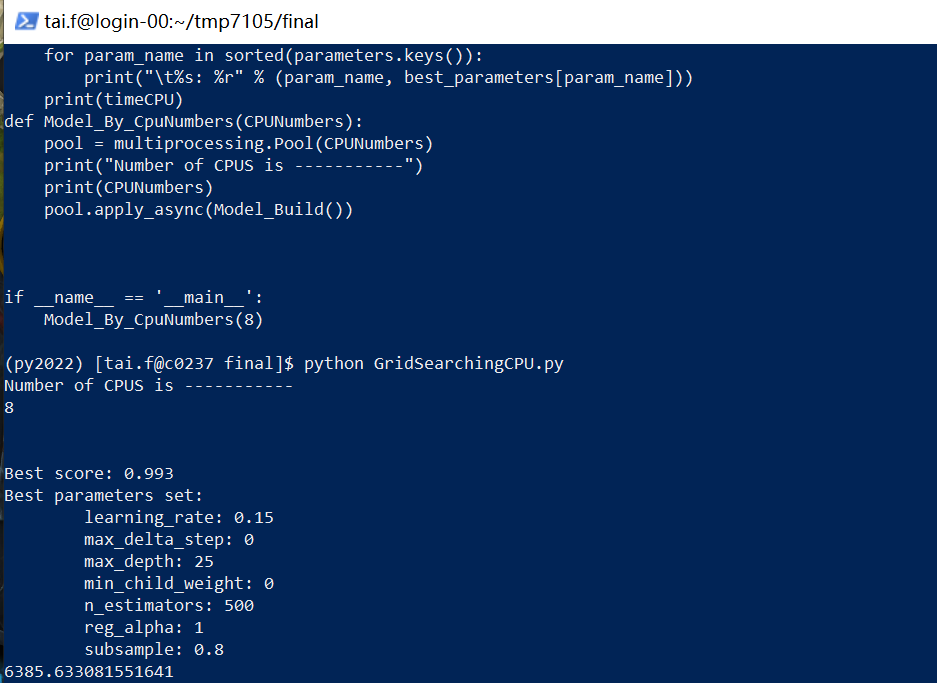


Figure 27

**Experiment 11: CPU = 1**

**GPU: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz**

**Reservation:** csye7105

**Method:** Dask, MultiProcessing



Figure 28

**Experiment 12: CPU = 2**

**GPU: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz**

**Reservation:** csye7105

**Method:** Dask, MultiProcessing

文本

描述已自动生成

Figure 29

**Experiment 13: CPU = 4**

**GPU: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz**

**Reservation:** csye7105

**Method:** Dask, MultiProcessing

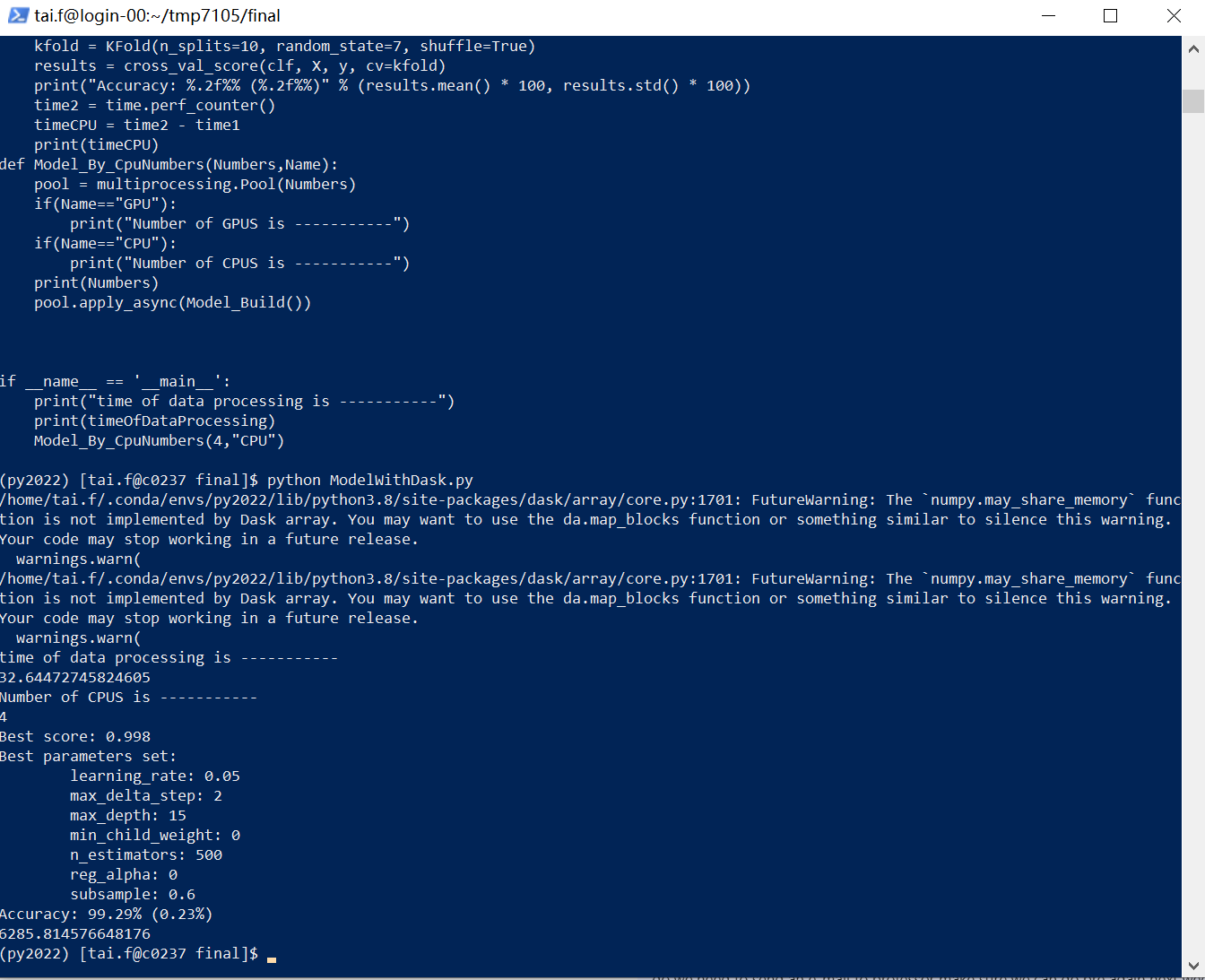


Figure 30

**Experiment 14: CPU = 8**

**GPU: Intel(R) Xeon(R) CPU E5-2690 v3 @ 2.60GHz**

**Reservation:** csye7105

**Method:** Dask, MultiProcessing

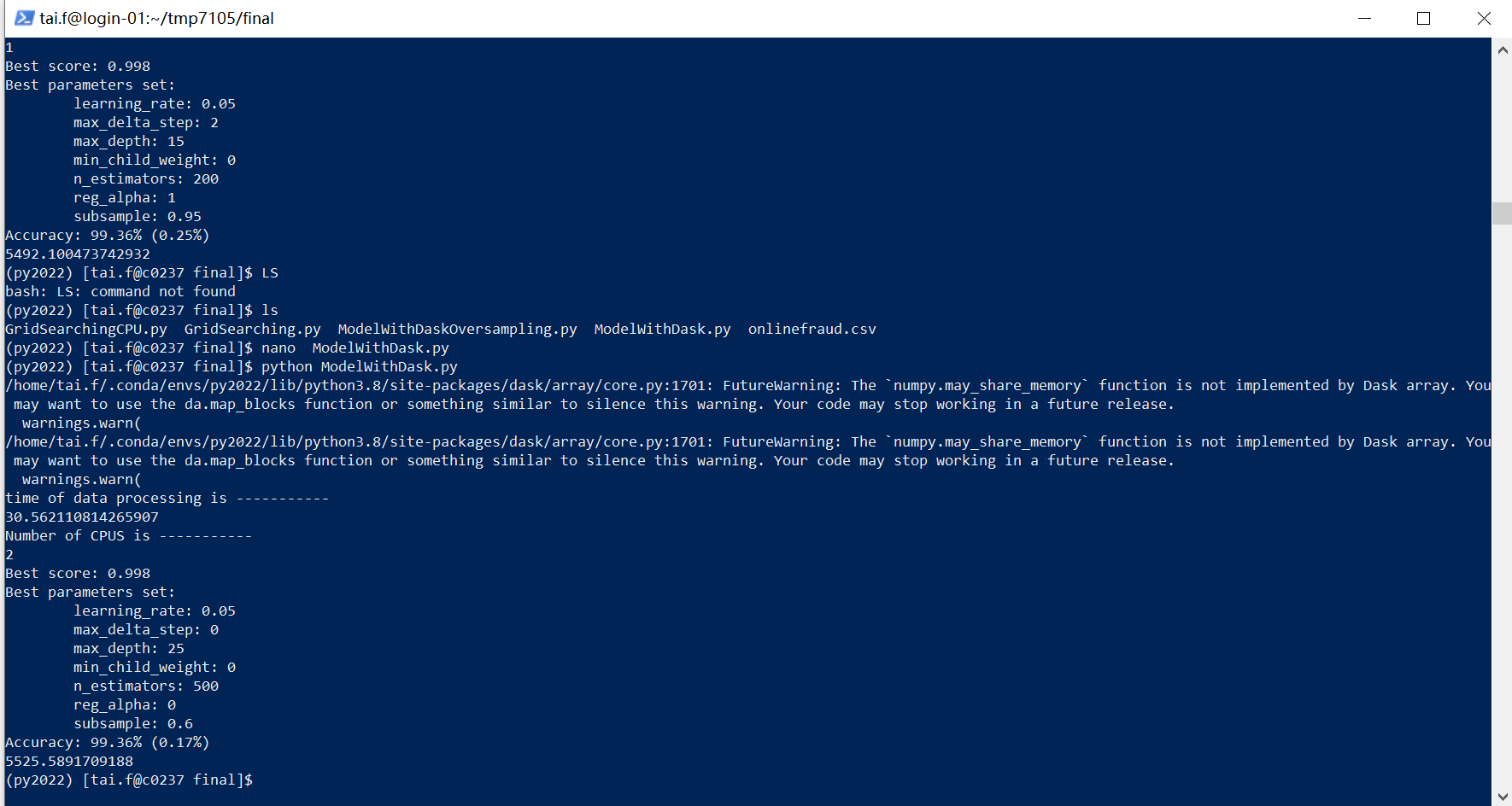


Figure 31

The result statistics are shown in the table below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Number of GPU or CPU | GPU or CPU | Execution Time  (Seconds) | Train Accuracy | Valid Accuracy | Method |
| 1 | GPU | 6529 | 0.9790665751544269 | 0.9726266744321491 | Pandas, MultiProcessing |
| 2 | GPU | 6494 | 0.9790665751544269 | 0.9726266744321491 | Pandas, MultiProcessing |
| 4 | GPU | 5867 | 0.9790665751544269 | 0.9726266744321491 | Pandas, MultiProcessing |
| 1 | GPU | 5808 | 0.9692832764505119 | 0.9710319855159928 | Dask, MultiProcessing |
| 2 | GPU | 5799 | 0.9706484641638226 | 0.9688436189334931 | Dask, MultiProcessing |
| 4 | GPU | 5099 | 0.9757679180887372 | 0.9674556213017751 | Dask, MultiProcessing |
| 1 | CPU | 6278 | 0.9783802333562114 | 0.9720442632498544 | Pandas, MultiProcessing |
| 2 | CPU | 6337 | 0.9790665751544269 | 0.9726266744321491 | Pandas, MultiProcessing |
| 4 | CPU | 6296 | 0.9783802333562114 | 0.9742539496781744 | Pandas, MultiProcessing |
| 8 | CPU | 6386 | 0.9790665751544269 | 0.9737302977232924 | Pandas, MultiProcessing |
| 1 | CPU | 5492 | 0.9757679180887372 | 0.9674556213017751 | Dask, MultiProcessing |
| 2 | CPU | 5526 | 0.9689419795221843 | 0.9710144927536232 | Dask, MultiProcessing |
| 4 | CPU | 6286 | 0.9696245733788396 | 0.971049457177322 | Dask, MultiProcessing |
| 8 | CPU | 6600 | 0.9689419795221843 | 0.9704463208685162 | Dask, MultiProcessing |

Obviously, the shortest running time is computed in parallel with Dask and multiprocessing on 4 GPUs. The longest running time was computed in parallel with Dask and multiprocessing on 8 CPUs.

**4.3.4 Performance of Speedup and Efficiency**

**CPU(Pandas):**

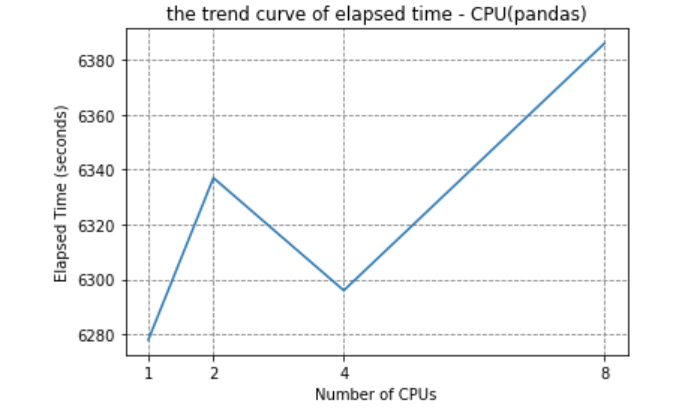


Figure 32

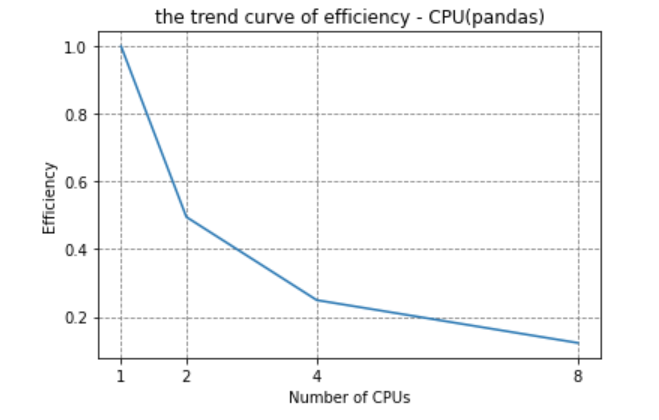
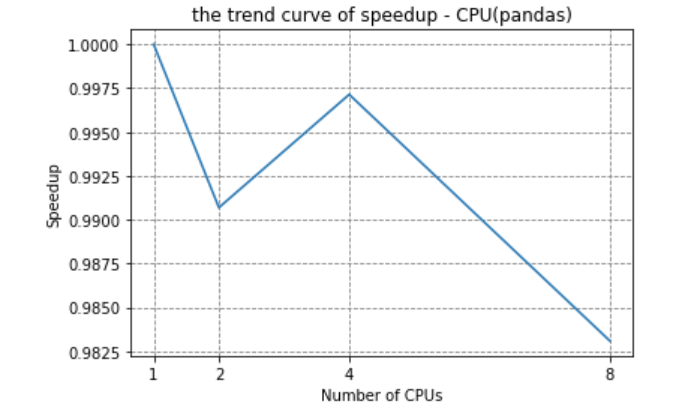


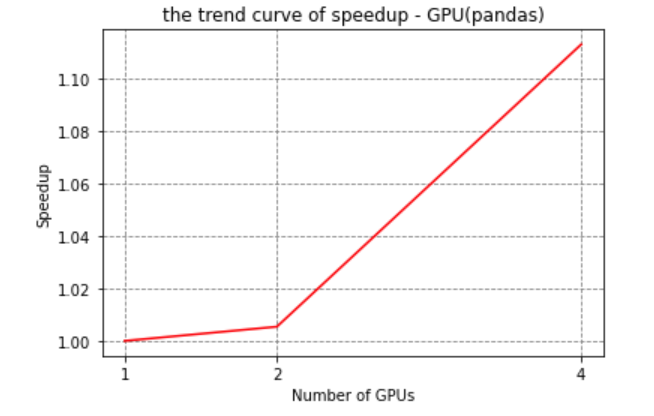
Figure 33 Figure 34

**GPU(Pandas):**

图表, 折线图

描述已自动生成

Figure 35

图表, 折线图

描述已自动生成

Figure 36 Figure 37

**CPU(Dask):**

图表, 折线图

描述已自动生成

Figure 38

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

Figure 39 Figure 40

**GPU(Dask):**

表格, 折线图

描述已自动生成

Figure 41

图表, 折线图

描述已自动生成图表, 折线图

描述已自动生成

Figure 42 Figure 43

According to the plot, the efficiency of all experiments shows a decreasing trend with the increase of the number of parallel cores, indicating that our parallel method may not be able to make good use of CPU and GPU resources. So, our optimal results only considering the running speed is to use Dask and multiprocessing to run on 4 GPU. And only in the case of using Dask and multiprocessing method on multiple-gpu, the speed up has an obvious upward trend. All other parallel acceleration effects are not obvious or even decline, especially when the number of CPUS is more than 4. But in general, the GPU speedup is much better than the CPU, and the overall speedup of Dask is also much better than pandas.

**4.3.5 Analysis of poor performance of Speedup and Efficiency**

* It's not always guaranteed that there will be preemption when run the program using all the resources.
* Some parts of model training or parameter adjusting, or cross-validation do not support Dask or multiprocessing for parallel processing.
* For the data processing itself, the CPU of a single process can handle it well and does not need additional parallel operations. Parallel operations themselves will bring more resource consumption and computing time
* Some data processing inevitably uses a mix of pandas and Dask, because the latest Dask does not fully support some pandas operations. Although the local impact of this time is very small, it may affect the overall effect of Dask lazy calculation.
* The compute () method is repeatedly called, which may have a performance poor effect.
* Do not adjust the chunk size to an optimal value.

**5. Conclusions**

* When using multiprocessing to parallelize a function, the cost of overhead needs to be considered, otherwise it may not be as effective as serial execution.
* Proper use of Dask Array can significantly improve the computation time of NumPy Array.
* Optimal results only considering the running speed is to use Dask and multiprocessing to run on 4 GPU because the whole efficiency is low.
* For our model parallelism is better on multiple-gpu than on multiple-cpu.
* The speedup of Dask is good on both multiple-gpu and multiple-gpu but the speedup of multiple-gpu is more significant

**6. References**

1. Fadyesam. “Online Fraud EDA and Prediction with Accuracy 99%.” *Kaggle*, Kaggle, 30 Oct. 2022, https://www.kaggle.com/code/fadyesam/online-fraud-eda-and-prediction-with-accuracy-99.

2. guest\_blog. “Imbalanced Classification: Handling Imbalanced Data Using Python.” *Analytics Vidhya*, 30 Nov. 2022, https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/.

3. Bahgat, Karim, et al. “Multiprocessing.pool() Slower than Just Using Ordinary Functions.” *Stack Overflow*, 1 Feb. 1961, https://stackoverflow.com/questions/20727375/multiprocessing-pool-slower-than-just-using-ordinary-functions.

4. Radečić, Dario. “Dask Arrays - How to Parallelize Numpy with Ease.” *Medium*, Towards Data Science, 23 Mar. 2022, https://towardsdatascience.com/dask-arrays-how-to-parallelize-numpy-with-ease-b33d7e9dcb59#:~:text=That's%20where%20Dask%20arrays%20provide,deal%2Dbreaker%20for%20some%20cases.

**5.** N'Wuitcha, Badji. “How Is Multiprocessing Different from Parallel Computing in Python?” *Medium*, Better Programming, 22 Oct. 2022, https://betterprogramming.pub/how-is-multiprocessing-different-from-parallel-computing-in-python-4b375c29b183.

6. Dhariwal, Kunal. “Pandas with Dask, for an Ultra-Fast Notebook.” *Medium*, Towards Data Science, 31 Oct. 2021, https://towardsdatascience.com/pandas-with-dask-for-an-ultra-fast-notebook-e2621c3769f.

7. Sravanti TatirajuI have been in the IT industry from 9 years. I worked as a curriculum validation engineer at Oracle for the past 5 years validating various courses on products developed by them. Before Oracle. “Optimize Running Large Number of Tasks Using Dask.” *Qxf2 BLOG*, 23 Mar. 2021, https://qxf2.com/blog/optimize-running-large-number-of-tasks-using-dask/.

8. Cavanaugh, Gus. “Quick and Dirty DataFrame Processing with CPU & GPU Clusters in the Cloud.” *Medium*, Medium, 29 Oct. 2021, https://medium.com/@GusCavanaugh/quick-and-dirty-dataframe-processing-with-cpu-gpu-clusters-in-the-cloud-c50d0ecfba92.