MiniProject 4: Reproducing "XGBoost" Scalable GPU Accelerated Learning

Fynn Schmitt-Ulms
McGill University

Joon Hwan Hong
McGill University

Daniel Korsunsky
McGill University

Reproducibility Summary

Scope of Reproducibility

In this paper, we reproduce Mitchell et al. (2018)'s findings that a multi-GPU-based extension of the XGBoost algorithm significantly decreases the algorithms runtime without affecting its accuracy by implementing prediction and other boosting elements all on the GPU (1). The authors claim that the GPU-enhanced XGBoost algorithm's runtime never exceeds two minutes on any dataset, is at least 7x faster than the CPU XGBoost, and maintains strong predictive performance for large-scale tasks. The objective of the reproduction is to verify the comparative speed of the GPU-enhanced XGB and validity of the authors' results.

Methodology

The five models we evaluated in our study were XGBoost (cpu and gpu version), LightGBM (cpu), and CatBoost (cpu and gpu). Our focus was on evaluating the runtime of the gpu implementation of XGBoost to other similar algorithms. We primarily used the authors' implementation (through the XGBoost python package) in our study. In particular, the authors provided code to run the benchmarks shown in their paper. We ran our computations on a single k80 GPU on Google Colab.

Results

Our results reproduced the original paper's claim that the GPU-enhanced XGBoost never exceeded two minutes on the large datasets, was at least 7x faster than the CPU XGBoost, and maintained strong predictive performance for large-scale tasks as the number of instances increased. We also verified that implementing prediction on the GPU instead of the CPU greatly contributed to increasing the model's runtime speed.

What was easy

The authors did a good job of making sure that their paper was reproducible, releasing a github repository (2) containing the benchmark tests they ran along with the hyperparameter settings used. We were able to use the authors' code with only minor modifications. There was also no issue obtaining the datasets and ensuring they were in the same format when passed into the models.

What was difficult

Our main difficulties involved computational limitations. For example, we omitted two datasets used in the original paper because they required too much memory (the authors suggested using a VM with 150 GB RAM). Also, the authors' implementation was provided as part of the XGBoost package which includes several other related algorithms, making the code base very large and thus difficult to understand and work with.

Communication with original authors

Due to the limited time for the assignment, we were unable to get in touch with the original authors before the assignment submission deadline.

Preprint. Under review.

1 Introduction

Gradient boosting is an ensemble algorithm applied to supervised learning tasks such as classification and regression, primarily to reduce model bias. Within boosting, XGBoost (extreme gradient boosting) is a specific implementation applied to classification and regression trees, and has extremely high accuracy, can handle sparse data, and is optimised for multicore implementation. Mitchell et al. (2018) extend the XGBoost library to enable multi-GPU-based execution, use less memory, and greatly decrease the runtime of large-scale tasks. The work achieves these results by implementing feature quantile generation, decision tree generation, prediction, gradient calculation, and evaluation algorithms on the GPU, accelerating the gradient boosting pipeline (1).

2 Scope of reproducibility

Mitchell et al. (2018)'s main contribution to the XGBoost library is a multi-GPU-based extension of the XGBoost algorithm which attempts to significantly decrease the algorithm's runtime without affecting its accuracy. By implementing feature quantile generation, decision tree generation, prediction, gradient calculation, and evaluation algorithms all on the GPU, the authors demonstrate that the algorithm is just as accurate but runs much faster than the standard CPU algorithm. They also evaluate their algorithm against the CPU and GPU versions of LightGBM (3) and CatBoost (4) on the YearPredictionMSD (5), Synthetic (6), Higgs (7), Cover Type (8), Bosch (9), and airline (10) datasets.

Due to computational limitations, our reproduction had to feature the following constraints:

- Because we are using Google's Colaboratory, we only have access to one k80 GPU, compared to Mitchell et al. (2018)'s maximum of 8 V100 GPUs.
- We did not attempt to reproduce results for the Airline and Bosch datasets because they were too big to fit into memory.
- We shrunk down the Synthetic dataset from 10M to 2M datapoints due to memory limitations.
- We had to omit LightGBM-GPU because we were unable to get it working in Colab.

Adjusting for these constraints, we have organized the original paper's results into 4 claims, which we will evaluate in Experiment 0:

- Claim 1: The GPU-enhanced XGBoost algorithm's runtime never exceeds two minutes on any dataset.
- Claim 2: The GPU-enhanced XGBoost algorithm is at least 7x faster (MSD) than the standard XGBoost (and up to 33x faster in the Cover Type dataset).
- Claim 3: The GPU-enhanced XGBoost algorithm is the fastest on 1/4 datasets (Cover Type).
- Claim 4: The GPU-enhanced XGBoost algorithm is the most accurate on 2/4 datasets (Higgs and Cover Type).

Additionally, while the original paper made additional claims, it did not provide data for them. Thus, we will analyze the following claims in additional experiments:

- Claim 5: The GPU-enhanced XGBoost algorithm significantly reduces the runtime of classification and regression tasks across a variety of dataset sizes, while maintaining strong predictive performance.
- Claim 6: Moving prediction onto the GPU significantly decreases algorithm runtime.

3 Methodology

We primarily used the authors' implementation (through the xgboost python package) in our study. In particular, the authors provided code to run the benchmarks shown in their paper. We ran all of our computations on a single K80 gpu on Google Colab.

3.1 Model descriptions

The five models we evaluated in our study were XGBoost (cpu and gpu version), LightGBM (cpu version), and CatBoost (cpu and gpu version). Our focus was on evaluating the runtime of the gpu implementation of XGBoost to other similar algorithms. To this end, all of the above algorithms are gradient boosting tree based models.

3.2 Datasets

The datasets used in our analysis are listed below.

Table 1: Datasets

Name	Instances	Features	Task
YearPredictionMSD(5)	515K	90	Regression
Synthetic(6)	2M	100	Regression
Higgs(7)	11M	28	Classification
Cover Type(8)	581K	54	Multiclass classification

Each dataset was split into train/val/test sets (60%/20%/20%). The Synthetic dataset, which we used for additional experiments, was generated using the sklearn "make_regression" function, which feeds random input data into a regression model and then adds gaussian noise to the output. For Experiment 2, the synthetic dataset we use is limited to only 500k data points.

3.3 Hyperparameters

We used the same hyperparameter settings as in the original paper for consistency. These are as follows:

- General hyperparameters: max depth: 6, learning rate: 0.1, min split loss: 0, min weight: 1, L1 regularization: 0, L2 regularization: 1
- LightGBM-specific hyperparameters: num_leaves: 2^8, min_data_in_leaf: 0

3.4 Experimental setup and code

For Experiment 0, we used the script provided by the paper authors to run the experiment. For the other experiments we made modifications to the synthetic dataset generating function so that it could accept the number of features and datapoints as input. For experiment 3 we made use of the "predictor" hyperparameter in the xgboost package that allowed the cpu to be set as the prediction device. We evaluated the results of our experiments using their running times. RMSE for regression datasets and accuracy for classification datasets was also recorded to ensure the models are performing well in their required tasks. Code provided with submission.

3.5 Computational requirements

Due to limited access to restricted GPU instances on Google Cloud, we decided to run our experiments entirely on the Google Colab platform. Thus utilizing a k80 GPU and 12 GB of RAM. All of the experiments together only took three hours to run.

4 Results

4.1 Results reproducing original paper

4.1.1 Experiment 0: Testing Claims

This experiment evaluated Claims 1-4, which corresponded to the algorithm's performance on the 4 datasets compared to the CPU-based XGBoost and the LightGBM and CatBoost models. We have included the authors' original results in *Table 1* for comparison to our own results in *Table 2*. As you can see in the table, Claim 1 was met by a large margin: the highest time was 71.5 seconds for the Cover Type dataset, which required multiclass classification on 581K instances and 54 features. Claim 2 was also met, with the GPU-enhanced algorithm outperforming the standard XGB by at least 7.5x (the Cover Type dataset) and as high as 16x more (the Synthetic dataset). Claim 3 was also successfully reproduced, with the GPU-enhanced XGB performing fastest on not only the Cover Type but also the MSD dataset. Finally, while Claim 4 was *not met* for any of the datasets, the GPU-enhanced XGB was very close in accuracy to the lightgbm-cpu algorithm (74.72 to 74.74).

4.2 Results beyond original paper

While the original authors supplied data for comparing the GPU-enhanced XGB against the standard XGB and other competing models, they did not provide evidence for other claims they had made, requiring additional experimentation

Table 2: Mitchell et al. (2018) Results

Model	MSD Time (s)	MSD RMSE	Synthetic Time (s)	Synthetic RMSE	Higgs Time (s)	Higgs Accuracy	Cover Type Time (s)	Cover Type Accuracy
xgb-cpu	216.71	8.8794	580.72	13.6105	509.29	74.74	3532.26	89.20
xgb-gpu	30.39	8.8799	43.14	13.4606	38.41	**74.75**	**107.70**	**89.34**
lightgbm-cpu	30.82	**8.8777**	463.79	13.5850	330.25	74.74	186.27	89.28
cat-cpu	39.93	8.9933	426.31	9.3870	393.21	74.06	306.17	85.14
cat-gpu	**10.15**	9.0637	**36.66**	**9.3805**	**30.37**	74.08	N/A	N/A

Table 3: Our Experiment 0 Results

Model	MSD Time (s)	MSD RMSE	Synthetic Time (s)	Synthetic RMSE	Higgs Time (s)	Higgs Accuracy	Cover Type Time (s)	Cover Type Accuracy
xgb-cpu-hist	119.12	8.8722	396.60	9.6102	771.47	74.73	539.74	89.32
xgb-gpu-hist	**8.22**	8.8775	22.03	9.6533	48.66	74.72	**71.50**	89.32
lightgbm-cpu	55.13	**8.8675**	207.30	9.7391	521.15	**74.74**	125.22	**92.23**
cat-cpu	63.95	8.9672	210.97	**5.6793**	1051.56	74.07	170.41	85.16
cat-gpu	8.26	9.2670	**16.97**	8.1305	**48.02**	74.06	N/A	N/A

on our end. We will be focusing on Claims 5 and 6 that the authors made: namely, that the GPU-enhanced XGB reduces the runtime of large-scale tasks (Experiments 1 and 2), and that moving prediction onto the GPU reduces algorithm runtime (Experiment 3).

For Experiments 1 and 2, we chose to experiment on the Synthetic dataset because, as it is synthetic, we could change the number of datapoints that got generated. To do the same with a real dataset, we'd have to choose a subset of the data, which might not be a representative sample and thus dilute the rigorousness of the experiment.

4.2.1 Experiment 1: Running Time as a Function of # of Datapoints

This and the following experiments evaluated the authors' claim that the algorithm was scalable and "significantly [reduced] the runtime of large-scale problems" (1). There is some evidence of this in Experiment 0, seeing that the xgb-gpu model was much faster the xgb-cpu model for all datasets, which are quite large (see Table 1). However, we wanted to control for the number of instances that the GPU model received as input to see if it improved against the CPU model as the number of instances increased. Indeed, as you can see in Figure 1(a), as the number of datapoints increases, the XGB-GPU model runtime barely increases (and remains consistently lower than the CPU model). The CPU model, on the other hand, performs significantly worse as the number of datapoints increases. Thus, we conclude that these data support the authors' claim.

For comparison, we also included the CatBoost model (with GPU and CPU versions) in Figure 1(b). As you can see, the models outperform their XGB counterparts but maintain similar trends, with the GPU model scaling well and running consistently faster than the CPU model, which performs worse as the number of datapoints increases.

4.2.2 Experiment 2: Running Time as a Function of # of Features

In this experiment, similar to Experiment 1, we evaluated the authors' claim that the algorithm was scalable and "significantly [reduced] the runtime of large-scale problems" (1), this time controlling for the number of features in the Synthetic dataset to see if the GPU model improved against the CPU model as the number of instances increased. Indeed, you can see in Figure 2(a), as the number of features increases (from 20 to 50 to 100 to 200), the XGB-GPU model runtime barely increases and remains consistently lower than the CPU model. The CPU model, on the other hand, performs significantly worse as the number of features increases. Thus, we conclude again that these data support the authors' claim.

For comparison, we also included the CatBoost model (with GPU and CPU versions) in Figure 2(b). As you can see, the models outperform their XGB counterparts but maintain similar trends, with the GPU model scaling well and running consistently faster than the CPU model, which performs worse as the number of features increases.

Figure 1: Experiment 1 Model Results

(a) XGBoost

(b) CatBoost

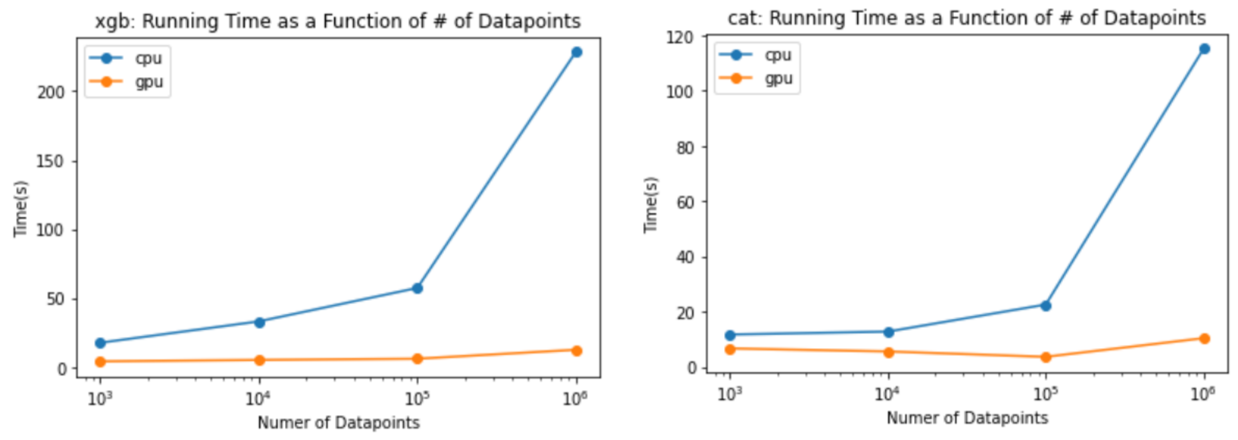
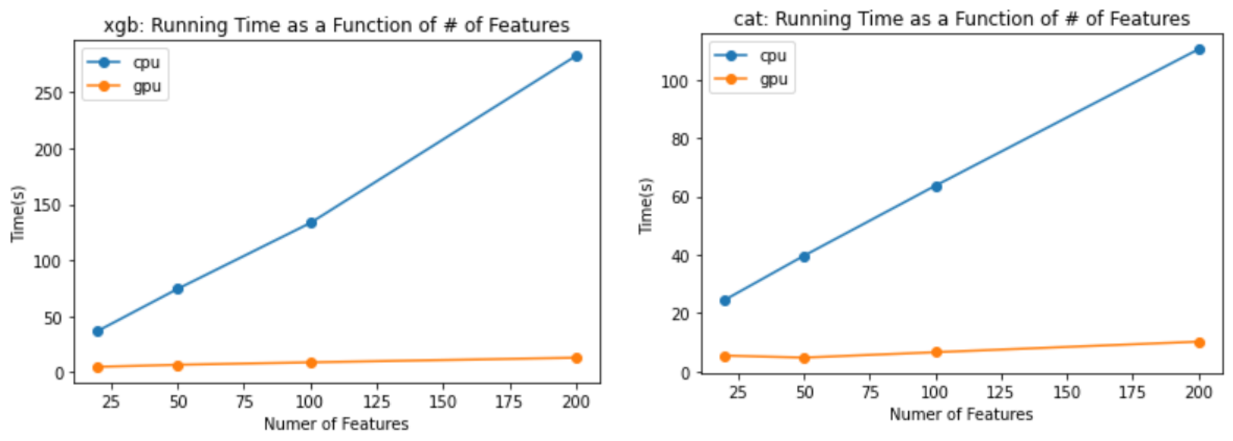


Figure 2: Experiment 2 Model Results

(a) XGBoost

(b) CatBoost



4.2.3 Experiment 3: Prediction Ablation

This experiment evaluated the authors' claim that implementing *prediction* to the GPU (as part of the overall enhanced XGB algorithm) contributed to decrease in runtime. The original paper only described this enhancement and did not provide data to justify it. Thus, we implemented an ablation experiment which forced the enhanced XGB algorithm to make prediction on the CPU rather than the GPU to see if runtime was affected. The authors also implemented other aspects of the algorithm on the GPU, such as feature quantile generation, decision tree generation, and gradient calculation. However, due to the confusing nature of the large XGBoost code base (see Section 5.2 for difficulties), we found that performing ablation on prediction was the most accessible option.

As you can see in *Table 3*, when we forced the GPU-enhanced XGB to make prediction on the CPU instead (xgb-gpu-hist-cpu-pred), runtime significantly increased, supporting the original authors' findings that implementing prediction on the GPU greatly reduced runtime. Note that the accuracies between the two models remained the same, or at least very similar.

Table 4: Experiment 3 Prediction Results

Model	MSD Time (s)	MSD RMSE	Synthetic Time (s)	Synthetic RMSE	Higgs Time (s)	Higgs Accuracy	Cover Type Time (s)	Cover Type Accuracy
xgb-gpu-hist-cpu-pred	28.07	8.8775	93.10	9.6533	201.19	74.72	369.19	89.32
xgb-gpu-hist	8.27	8.8775	21.32	9.6846	47.17	74.74	69.14	89.32

5 Discussion

We obtained very similar results in our reproduction of the author's experiments (in Experiment 0): the GPU implementation of XGBoost had significantly reduced runtime relative to the CPU version while achieving similar predictive performance.

We found in Experiments 1 and 2 that this result is robust to variations in the number of input features as well as the size of the datasets used. Therefore, the results from the authors' paper are not specific to the specifications of the datasets analyzed but rather are true across a broader spectrum of inputs.

Through Experiment 3, we showed that the authors' decision to move prediction onto the GPU was justified, as removing this feature greatly increased the runtime of training the model.

Thus, we conclude that the authors' results were not only reproducible for the data they used but appear to be robust to reasonable perturbations.

5.1 What was easy

The authors did a good job of making sure that their paper was reproducible. Alongside their paper, they released a github repository (2) containing the benchmark tests they ran and along with all the hyperparameter settings used. In addition there were instructions as to which kind of cloud instance they recommend using for the benchmarks. We were able to primarily use the authors' code to run their experiments with only minor modifications.

In addition, the authors' code included scripts for downloading and prepossessing the datasets they used, all of which are publicly available. Therefore, there was no issue obtaining the datasets and ensuring they were in the same format when passed into the models.

5.2 What was difficult

Our main difficulties in implementing the paper were due to computational limitations. We had to modify the synthetic dataset to use less data points, due to limited RAM. In addition we were unable to run experiments on two of the six datasets in the original paper because they required too much memory (the authors suggested using a VM with 150 GB RAM).

In addition, the authors' implementation of the GPU version of the algorithm was only provided as part of the XGBoost package which includes several other related algorithms and support for several other languages. This makes the code base very large, and therefore difficult to understand and work with. In particular, there was no easy way to see the code used specifically for the GPU addition described in the paper. This made it difficult to directly observe the way they implemented the algorithm and to modify it for ablation studies.

5.3 Communication with original authors

In an ideal scenario, we would have gotten in touch with the original authors to ask questions and make sure our report was a reasonable assessment of their work. However, due to the limited time, we were unable to do so before the assignment submission deadline.

6 Statement of Contributions

Contributions by Schmitt-Ulms: Experiment 0 & 3, Report writing.
Contributions by Hong: Experiment 1 & 2, Report editing.
Contributions by Korsunsky: Experiment 1 & 2, Report writing.

References

[1] Mitchell, Rory & Adinets, Andrey & Rao, Thejaswi & Frank, Eibe. (2018). XGBoost: Scalable GPU Accelerated Learning.

[2] Mitchell, R (2018) GBM-Benchmarks [Source code]. https://github.com/RAMitchell/GBM-Benchmarks.

[3] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, andTie-Yan Liu. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *In Advances in Neural Information Processing Systems*, p.3149–3157.

[4] Anna Veronika Dorogush, Andrey Gulin, Gleb Gusev, Nikita Kazeev, Liudmila Ostroumova Prokhorenkova, and Aleksandr Vorobev. (2017). Fighting biases with dynamic boosting. *CoRR*, abs/1706.09516.

[5] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011), 2011.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12:2825–2830, 2011.

[7] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high- energy physics with deep learning. Nature communications, 5:4308, 2014.

[8] Jock A. Blackard. Comparison of Neural Networks and Discriminant Analysis in Predicting Forest Cover Types. PhD thesis, Fort Collins, CO, USA, 1998. AAI9921979.

[9] Kaggle bosch production line performance dataset. https://www.kaggle.com/c/bosch- production-line-performance/data.

[10] Airline dataset. http://kt.ijs.si/elena_ikonomovska/data.html.