# AFNFA: An Approach to Automate NCCL Configuration Exploration

Zibo Wang[★], Yuhang Zhou[★], Chen Tian[★], Xiaoliang Wang[★], Xianping Chen[△]
[★]*Nanjing University, China* [△]*Huawei, China*

## ABSTRACT

With the continuously increasing scale of deep neural network models, there is a clear trend towards distributed DNN model training. State-of-the-art training frameworks support this approach using collective communication libraries such as NCCL, MPI, Gloo, and Horovod. These libraries have many parameters that can be adjusted to fit different hardware environments, and these parameters can greatly impact training performance. Therefore, careful tuning of parameters for each training environment is required. However, given the large parameter space, manual exploration can be time-consuming and laborious.

In this poster, we introduce AFNFA, which stands for **A**I **F**or **N**etwork **F**or **A**I. It is an automated program that utilizes machine learning and simulated annealing to explore NCCL parameters. Preliminary evaluation results demonstrate that compared to the default configuration, the configuration explored by AFNFA improves NCCL communication performance by 22.90%.

## 1 INTRODUCTION

In recent years, deep neural networks (DNNs) have been increasingly used in various industries. As the number of model parameters continues to increase, DNNs are capable of achieving better performance. To satisfy the ever growing computing power required for training increasingly large models, distributed machine learning has become more commonly adopted.

In distributed machine learning, communication is crucial because it involves data transfer among multiple GPUs or even multiple machines, which significantly impacts the overall training time. To facilitate efficient communication, current training frameworks rely on collective communication primitives, such as all-reduce and all-to-all, which are implemented in communication libraries. Among different communication libraries, NCCL[6] is the most widely used one because of its support for advanced features, such as InfiniBand, GPU Direct, NvLink, and NvSwitch.

However, available hardware is rapidly evolving, resulting in varying hardware environments for different users. NCCL has many parameters that users can configure to adapt to different hardware environments and improve the performance of data transfer to increase the training speed. We have conducted several tests on NCCL in a production environment, and the results show that different configurations have a significant impact on NCCL performance and need to be carefully tuned. However, given the large parameter space, manual exploration can be time-consuming and laborious.

To address this issue, we propose AFNFA, an automated program that employs machine learning and simulated annealing to explore NCCL configurations.We first train a regression model of NCCL performance with respect to configuration, and then use simulated annealing[1] to find the maximum value of the model, which is the predicted maximum performance, and the corresponding configuration, which can be used for all future training when the hardware environment remains unchanged. The results of our tests show that configuration explored by AFNFA yields a 22.90% improvement in performance compared to the default configuration.

## 2 DESIGN

Implementing an automated configuration exploration program involves more than just automating the exploration process. The time overhead would be unacceptable if the program were to automatically traverse the entire parameter space. Thus, the key is to narrow the search space in various ways, which helps reduce the number of explorations.

### 2.1 Overview

As machine learning can reveal hidden associations in vast amounts of data, we believe that applying it to this scenario can effectively reduce the number of explorations and decrease the time overhead of exploration.



*APNET 2023, June 29–30, 2023, Hong Kong, China*

ACM ISBN 979-8-4007-0782-7/23/06.
https://doi.org/10.1145/3600061.3600068

*2.1.1 Determining the configuration parameter space.* We conducted several tests in a production environment by manually altering the environment variables provided on the official NCCL website[1]. Subsequently, we selected the variables that demonstrated a substantial impact on performance.

- NCCL_MAX/MIN_NCHANNE
- NCCL_SHM_DISABLE
- NCCL_P2P_LEVEL
- NCCL_SOCKET_NTHREADS
- NCCL_BUFFSIZE
- NCCL_NET_GDR_LEVEL
- NCCL_ALGO

*2.1.2 Constructing the training dataset.* It is apparent that the solution space, which is the Cartesian product of the parameter space of each variable, is extremely vast, with up to 61920 possible combinations, making it unfeasible and impractical to test each one of them. In order to mitigate the time overhead of the process, we randomly sample 5% of variable combinations from the full set to execute NCCL-test, and record the resulting data. The final dataset contains 345 samples.

*2.1.3 Selecting model and training.* To predict the corresponding NCCL-test performance based on the selected configuration, we approach this as a regression problem. We use the dataset constructed in the previous step to train a function y=f(x), where x represents the input NCCL configuration and y represents the NCCL-test performance predicted by this function for that configuration.

Firstly, we need to select a suitable regression model for the task. We performed several tests with the commonly used regression algorithms, measuring the maximum value obtained by each algorithm and its error with the true value obtained from the production environment with the same configuration. The results are shown in the table 1. After carefully comparing the ease of implementation and the accuracy of the algorithm, we ultimately selected random forest regression as our regression model.

*2.1.4 Selecting the optimal values.* We use simulated annealing, a randomized selection algorithm to find the maximum value of the model obtained from the third training step and the corresponding input parameters. We use the predicted results of the model as the input parameters for the simulated annealing algorithm.

## 3 PRELIMINARY EVALUATION

We have implemented AFNFA in approximately 1000 lines of Python code, created the regression model using sklearn, and utilized MPI to automate running the test on multiple machines.

[1]https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/env.html

**Table 1: Performance of different model**

| Model | Predicted results | Errors |
|---|---|---|
| linear regression[8] | 0.2123 GB/s | 38.90% |
| ridge regression[4] | 0.2289 GB/s | 34.13% |
| MLP regression[2] | 0.3100 GB/s | 10.79% |
| random forest regression[3] | 0.3365 GB/s | 3.17% |
| Adaboost regression[7] | 0.3879 GB/s | 11.63% |

We performed a preliminary evaluation of AFNFA on a machine learning training platform provided by our university. Each machine in this platform is equipped with eight NVidia V100 GPUs connected by NVLink, while the different machines are connected to each other through a lower bandwidth Ethernet.

We conducted five NCCL-test runs for configuration generated by AFNFA, resulting in an average performance of 0.3475GB/s, which was a 22.90% improvement over the default configuration's average of 0.2828GB/s. We also evaluated the DLRM model[5] training time, and found that the configuration generated by AFNFA had a training time of 25ms per round, which was more than 10% faster than the default configuration's training time of 28ms per round.

## 4 CONCLUSION

In this poster, we propose AFNFA, an automated NCCL parameter exploration method using model training and simulated annealing, which we implement as a python program and can be used in any environment. Our preliminary evaluation results demonstrate that the configuration explored by AFNFA significantly improves NCCL communication performance by 22.90% compared to the default configuration.

## REFERENCES

[1] Dimitris Bertsimas and John Tsitsiklis. 1993. Simulated annealing. *Statistical science* 8, 1 (1993), 10–15.
[2] Simon Haykin. 1994. *Neural networks: a comprehensive foundation*. Prentice Hall PTR.
[3] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, Vol. 1. IEEE, 278–282.
[4] Gary C McDonald. 2009. Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics* 1, 1 (2009), 93–100.
[5] Maxim Naumov, Dheevatsa Mudigere, Hao-Jun Michael Shi, Jianyu Huang, Narayanan Sundaraman, Jongsoo Park, Xiaodong Wang, Udit Gupta, Carole-Jean Wu, Alisson G Azzolini, et al. 2019. Deep learning recommendation model for personalization and recommendation systems. *arXiv preprint arXiv:1906.00091* (2019).
[6] NVIDIA. [n. d.]. NVIDIA Collective Communications Library (NCCL). Website. ([n. d.]). https://developer.nvidia.com/nccl.
[7] Robert E Schapire. 2013. Explaining adaboost. In *Empirical inference*. Springer, 37–52.
[8] Xiaogang Su, Xin Yan, and Chih-Ling Tsai. 2012. Linear regression. *Wiley Interdisciplinary Reviews: Computational Statistics* 4, 3 (2012), 275–294.