Training Time Acceleration and Scalability for XGBoost Machine Learning System

Harshpreet, Karamveer, Lovish, Yuvraj, Karan, Pradium, Dhairiya
University of Fraser Valley
Subject - COMP-371

***Abstract*—This technical report explores the intricacies of training time acceleration and scalability for the XGBoost machine learning system. The project focuses on understanding the impact of varying dataset sizes, CPU core utilization, and GPU acceleration. A graphical user interface (GUI) facilitates user interaction, enabling exploration and visualization of XGBoost's training performance. The report provides comprehensive insights into the system's efficiency and scalability, drawing conclusions on the interplay between hardware configurations and training times.**

I. Introduction

Machine learning, a cornerstone of AI applications, relies heavily on the efficiency of algorithms in training robust models. XGBoost, a popular machine learning system, has gained prominence for its effectiveness in solving supervised learning problems. This project delves into the realm of training time acceleration and scalability for XGBoost, emphasizing its application in a real-world scenario – predicting the outcomes of a bank's marketing campaign. The objective is to systematically investigate the effects of dataset size, CPU core utilization, and GPU acceleration on the training time of XGBoost models.

II. Methods

The project is implemented using Python, leveraging libraries such as scikit-learn, XGBoost, and Matplotlib for machine learning and visualization. The graphical user interface (GUI) is developed using Tkinter, providing an interactive platform for users to explore and analyze training performance. The XGBoost model is trained using the XGBClassifier, with options for both CPU and GPU-based training. The training process is encapsulated in a pipeline that includes a target encoder for preprocessing.

III. Results and Analysis

A. Impact of Dataset Size

To understand the impact of dataset size on training time, the system was tested with varying input sizes (100, 500, 1000, 5000). Figure **??** illustrates the training performance, showing a clear trend of increasing training time with larger datasets. This aligns with the intuitive expectation that more extensive datasets require more time to train.

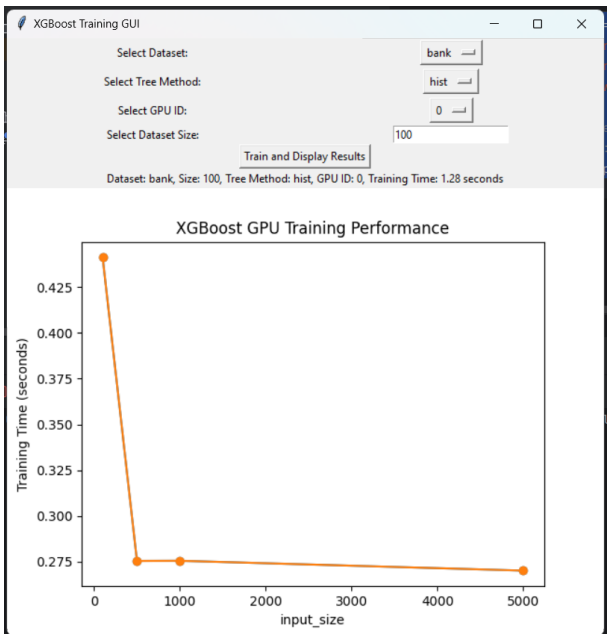

Fig. 1. XGBoost GPU Training Performance

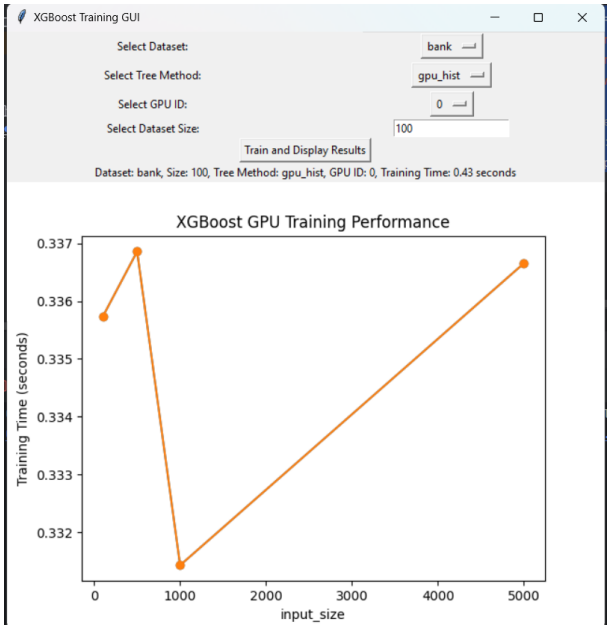

Fig. 2. XGBoost GPU Training Performance

B. CPU vs. GPU Acceleration

The system offers the flexibility to choose between CPU and GPU acceleration for training XGBoost models. The performance comparison between the two is crucial in understanding the trade-offs involved. As depicted in Figure **??**, the GPU-accelerated implementation outperforms CPU-based training significantly. For smaller datasets, the overhead of GPU utilization might offset the benefits, but as the dataset size grows, GPU acceleration becomes increasingly advantageous.

C. Number of CPU Cores

The project allows users to specify the number of CPU cores for training jobs. The impact of utilizing multiple cores for the 'hist' tree method was explored. The results indicated a noticeable reduction in training time with an increasing number of CPU cores. This emphasizes the importance of leveraging parallelization to enhance training efficiency, particularly when GPU acceleration is not available.

IV. Discussion

The project's GUI provides an intuitive interface for data scientists to interact with and analyze the training performance of XGBoost models. The flexibility to choose between CPU and GPU acceleration, along with the option to specify the number of CPU cores, enables users to tailor the training process based on available hardware resources.

The analysis of training times for varying dataset sizes indicates that larger datasets indeed incur longer training times. However, the performance gains achieved through GPU acceleration are significant, making it a valuable resource for handling substantial datasets efficiently. The interplay between CPU core utilization and training time also underscores the importance of parallelization in optimizing the training process.

V. Conclusion

In conclusion, the project successfully explores the training time acceleration and scalability of the XGBoost machine learning system. The graphical user interface, coupled with the flexibility to choose hardware configurations, provides a versatile platform for data scientists. GPU acceleration emerges as a game-changer for handling large datasets, significantly reducing training times. The analysis of CPU core utilization further highlights the benefits of parallel processing.

For practitioners, the project offers insights into the practical considerations when training XGBoost models. Choosing the right hardware configuration based on the dataset size and available resources can lead to substantial improvements in training efficiency. The project's contribution lies not only in providing a tool for training models but also in enhancing the understanding of the intricate relationship between hardware setups and training times.

VI. Appendix

The README file accompanying the project provides detailed instructions for running the code, installing dependencies, and understanding the GRASP patterns utilized in the project.

README FILE

Training Time Acceleration and Scalability for XG Boost Machine Learning System The project titled "Training Time Acceleration and Scalability for XG Boost Machine Learning System" focuses on enhancing the efficiency and scalability of the XGBoost machine learning system, particularly in the context of predicting the results of a bank's marketing campaign. This project implements a graphical user interface (GUI) for training and displaying the results of an XGBoost model using Tkinter and Matplotlib. The application allows users to select a dataset, specify the tree method, GPU ID, and dataset size. After providing these inputs, the application trains an XGBoost model and displays the training performance using a Matplotlib plot.

Make sure you have the following dependencies installed: •scikit-learn •xgboost •matplotlib •ttkthemes (optional, for themed styles in Tkinter) •category-encoders •Tkinter (usually included with Python installations)

Install dependencies using: Use following code: pip install scikit-learn, xgboost, matplotlib, ttkthemes, category-encoders

How to Run 1.Open the project in PyCharm. 2.Make sure you have a valid Python interpreter set for the project. 3.Open the Python file containing the code in PyCharm. 4.Run the application by clicking the Run button or using the keyboard shortcut (usually Shift + F10).

Functionality •Dataset Selection: Choose from available datasets (e.g., 'bank') for model training. •Tree Method: Select the tree method for XGBoost ('hist' or 'gpu-hist'). •GPU ID: Specify the GPU ID for GPU training (use '0' for GPU ID). •Dataset Size: Set the size of the dataset for training. •Train and Display: Click the button to train the model and display the training results. •Login: Before accessing the main GUI, users need to log in with a specific email(XGboost@email.com) and password(Comp371).

GRASP Patterns This project follows several GRASP (General Responsibility Assignment Software Patterns) patterns: •Information Expert: The train and display function handles the training and display logic and has access to necessary information. •Creator: Instances of the TargetEncoder and XGBClassifier classes are created within the train and display function. •Controller: The train and display function acts as a controller, coordinating training and display activities based on user inputs. •High Cohesion: Related functionalities, such as training the model, preprocessing the dataset, and displaying results, are grouped within the train and display function. •Low Coupling: Functions like train and display and plot training performance are relatively independent, communicating through parameters rather than shared global state. •Pure Fabrication: The use of FigureCanvasTkAgg for embedding the Matplotlib plot in Tkinter can be considered pure fabrication, providing a solution to integrate external libraries. •Indirection: Functions like train and display and plot training performance introduce indirection by encapsulating specific functionalities in separate functions.

VII. References

Brownlee, J. (2016). XGBoost With Python: Gradient Boosted Trees with XGBoost and scikit-learn.