EPiC Series in Computing

Volume 104, 2024, Pages 243–267

Proceedings of 3rd International Workshop on Mathematical Modeling and Scientific Computing


# Performance Analysis of Parallel Programs with RAPIDS as a Framework of Execution

Seyi T. Ogunji[1], Moises Sánchez-Adame[2], Oscar H. Montiel[3], and Juan J. Tapia[4]

[1] Instituto Politécnico Nacional, Mexico City, CITEDI, Mexico
seyitope@citedi.mx
[2] Instituto Politécnico Nacional, Mexico City, CITEDI, Mexico
mosancheza@ipn.mx
[3] Instituto Politécnico Nacional, Mexico City, CITEDI, Mexico
oross@ipn.mx
[4] Instituto Politécnico Nacional, Mexico City, CITEDI, Mexico
jtapiaa@ipn.mx

**Abstract**

In this age where data is growing at an astronomical rate, with unfettered access to digital information, complexities have been introduced to scientific computations, analysis, and inferences. This is because such data could not be easily processed with traditional approaches. However, with innovative designs brought to the fore by NVIDIA and other market players in recent times, there have been productions of state-of-the-art GPUs such as NVIDIA A100 Tensor Core GPU, Tesla V100, and NVIDIA H100 that seamlessly handle complex mathematical simulations and computations, artificial intelligence, machine learning, and high-performance computing, producing highly improved speed and efficiency, with room for scalability. These innovations have made it possible to efficiently deploy many parallel programming models like shared memory, distributed memory, data parallelization, and Partitioned Global Address Space (PGAS) with high-performance metrics. In this work, we analyzed the parquet-formatted New York City yellow taxi dataset on a RAPIDS and DASK supported distributed data-parallel training platform using a high-performance cluster of 7 NVIDIA TITAN RTX GPUs (24GB GDDR6 each) running CUDA 12.4. The dataset was used to train Extreme Gradient Boosting (XGBoost), RandomForest Regressor, and Elastic Net models for trip fare predictions. Our models achieved notable performance metrics. The XGBoost achieved a mean squared error (MSE) of 10.87, $R^2$ of 96.9%, and a training time of 21.1 seconds despite the huge size of the training dataset, showing how computationally efficient the system was. RandomForest achieved MSE of 27.46, $R^2$ of 92.2% and a training time of 25.9 seconds. In the bid to show the scalability and versatility of our experimental design to different machine learning domains, our multi-GPU accelerated training was extended to image classification tasks by using MobileNet-V3-Large pre-trained architecture on a CIFAR-100 dataset. The following parallelization results were achieved: a low Karp-Flatt metric of 0.013, indicating minimal serialization, 98.7% parallel fraction, demonstrating excellent parallelization, and only 7.1% communication overhead relative to computation time. For the model

V.L. Turova, A.E. Kovtanyuk and J. Zimmer (eds.), MMSC 2024 (EPiC Series in Computing, vol. 104), pp. 243–267

performance, we achieved a ROC_AUC of over 95% for the implementation. This work advances the state-of-the-art in parallel computing through implementation of RAPIDS and DASK frameworks on a distributed data-parallel training platform making use of NVIDIA multi-GPUs. The work is built on a well established theoretical framework using Amdahl and Gustafson's laws on parallel computation. By integrating RAPIDS and DASK, we contribute to advancing parallel computing capabilities, offering potential applications in smart city development and the field of logistics and transportation management services where rapid fare predictions are very important. The contribution could also be extended to the field of image classification, vision systems, object detection and embedded systems for mobile applications.

# 1 Introduction

The field of scientific computing is replete with different GPU-powered parallel programming models because they are very inevitable in the efficient processing of large datasets that are available in real-world use cases. They have a mathematical framework on Amdahl and Gustafson's equations [2, 4, 8] which are expressed as:

$$S(n) = \frac{1}{1 - p + p/n} \tag{1}$$

and

$$S(n) = n - \alpha(n - 1), \tag{2}$$

where

- $S(n)$ is the speedup achieved using $n$ processors,
- $p$ is the parallelizable portion of the program,
- $1 - p$ is the sequential (non-parallelizable) portion of the program,
- $n$ is the number of processors,
- $\alpha$ is the fraction of time spent on the serial (non-parallelizable) portion of the program,
- $n - 1$ is the number of parallel workers relative to the serial execution.

They provide conceptual frameworks that influence the logical paths employed by programmers either during application or algorithm design or when fixing bugs in the system. The models are multifaceted: ranging from shared memory, distributed memory, hybrid model, data-parallel model, and many more. The focus of this work is on the use of RAPIDS, which is an open-source library that unlocks the speed of GPU for cost-effective computations [22]. This open-source library is based on the data parallel model. When used within a single GPU, it uses shared memory parallelism. It is built on the CUDA programming model [22]. Even though RAPIDS itself does not implement distributed memory parallelism, as shown in this work, it could be used alongside frameworks such as DASK for distributed computing across several nodes and multiple GPUs.

The remaining part of this paper is structured as follows: Section 2 describes the methodologies that were used for the experiments. Section 3 addresses the results with their detailed explanations. Section 4 discusses the conclusion inferred from the work and the focus for future work.

# 2 Methodology

This work demonstrates sophisticated pipelines used in the two experiments where the RAPIDS environment was used in conjunction with the DASK framework to enhance proper implementations of distributed data-parallel training. The work of [24] revealed the evolution of parallel computing and talked about data parallelism, model parallelism, and pipeline parallelism; we termed these DMP parallelisms. The methodology used in this work focused more on the data and pipeline parallelisms where the DASK framework was used in conjunction with RAPIDS for parallelism of data and pipelines across multiple GPUs. Beyond the theoretical framework in the literature, we provided some empirical data to show the viability of these approaches.

Each of these experiments is analyzed below with their corresponding pipeline.

## 2.1 Experiment 1: Distributed data-parallel training of New York City taxi dataset using XGBoost, Elastic Net and Random Forest with RAPIDS integration

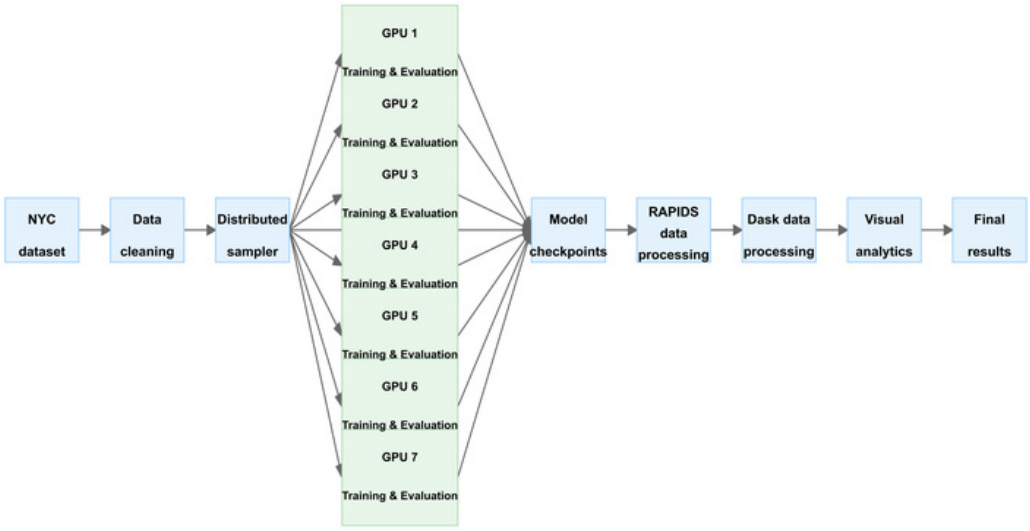



Figure 1: RAPIDS_DASK distributed training pipeline for NYC taxi data.

In the first experiment (see Fig. 1), the popular New York City yellow taxi dataset [18], with records of 3,539,193 rows and 19 columns for June 2024, was used and trained with traditional machine learning algorithms comprising XGBoost [5], that relies on this mathematical framework:

$$\text{Obj}(\theta) = L(\theta) + \Omega(\theta), \tag{3}$$

where

- $\text{Obj}(\theta)$ is the objective function to minimize during training,

- $L(\theta)$ is the loss function, representing the error of predictions,
- $\Omega(\theta)$ is the regularization term to control model complexity.

Random Forest [3], mathematically represented by the equation:

$$\hat{y} = \frac{1}{M} \sum_{m=1}^{M} f_m(x), \tag{4}$$

where

- $\hat{y}$ is the predicted value,
- $M$ is the number of decision trees in the random forest,
- $f_m(x)$ is the prediction from the $m$-th decision tree.

The Elastic Net [25], architecturally described by this mathematical expression:

$$L(\boldsymbol{\beta}) = \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2 + \lambda_1 \|\boldsymbol{\beta}\|_1 + \lambda_2 \|\boldsymbol{\beta}\|_2^2, \tag{5}$$

where

- $L(\boldsymbol{\beta})$ is the loss function,
- $\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$ is the residual sum of squares,
- $\lambda_1 \|\boldsymbol{\beta}\|_1$ is the Lasso (L1) regularization term,
- $\lambda_2 \|\boldsymbol{\beta}\|_2^2$ is the Ridge (L2) regularization term.

XGBoost has proven to be a reliable algorithm in this type of predictive task as shown by M. Poongodi et al. [20] in their work. In our work, we were able to achieve above 99% for predictions of some price ranges. The New York City yellow taxi dataset has been used widely in various machine learning applications, particularly for fare prediction and trip duration estimation like in many other works where taxi-related datasets have been used [6, 14].

By taking advantage of cuPy, cuML, and cuDF libraries of the RAPIDS framework, the models were trained over the NVIDIA GPU system taking advantage of the recent revolutionary trends in scalable general-purpose GPU computing [23]. The model was later used to make fare predictions. The models' performances in terms of parallelism metrics, selected error metrics, and fare predictions were tracked for visualizations.

## 2.2 Experiment 2: Distributed data-parallel training of CIFAR-100 on MobileNet-V3-Large using PyTorch and DASK with RAPIDS

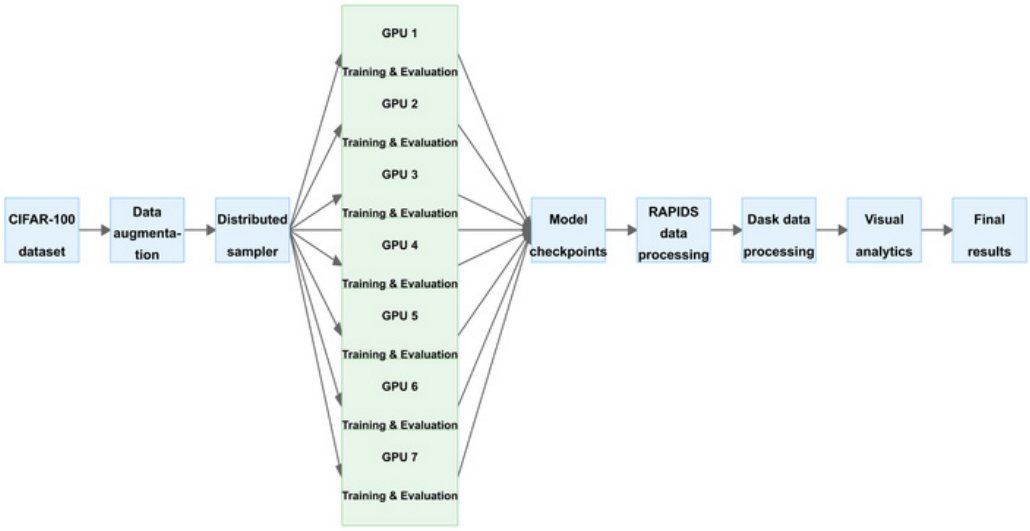



Figure 2: RAPIDS_DASK distributed training pipeline.

This setup (see Fig. 2) utilizes some of the libraries that are provided by RAPIDS. They include cuDF for GPU-accelerated data frame manipulations and computations, cuPy for GPU-accelerated array operations, and cuML for rigorous implementation of deep learning architecture [22]. The methodology further incorporates DASK for optimum parallelism of the data across multi-GPUs, which in this experiment were seven. The speed-up gain [10] could be seen in this expression:

$$S = \frac{T_1}{T_p}, \tag{6}$$

where

- $S$ is the speedup due to parallelization,
- $T_1$ is the execution time of the sequential algorithm,
- $T_p$ is the execution time using $p$ processors.

The PyTorch library was further used to complement the DASK framework for better acceleration of the task and model training purpose [19]. The architecture used for the experiment was a pre-trained MobileNet-V3-Large [12], which was further fine-tuned before the final classification layer with the introduction of 512 neurons and a dropout normalization strategy to enhance better model performance and generalization [11]. For the final classification into classes, the softmax [7] applied to the logits during the loss calculation is used since it has been internally incorporated with the CrossEntropyLoss function [16].

The probability output for the $i$-th class is defined as:

$$\sigma(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_j \exp(z_j)}, \tag{7}$$

where

- $\sigma(\mathbf{z})_i$ is the probability output for the $i$-th class,
- $z_i$ is the input score for the $i$-th class,
- $\sum_j \exp(z_j)$ is the sum of exponential scores for all classes.

The cross-entropy loss function is then used to measure the difference between the predicted probabilities and the true distribution by using the equation:

$$H(p, q) = -\sum_x p(x) \log(q(x)), \tag{8}$$

where

- $H(p, q)$ is the cross-entropy loss,
- $p(x)$ is the true probability distribution,
- $q(x)$ is the predicted probability distribution.

A series of data augmentation techniques such as random crop, random horizontal flip, and color jitter were used on the architecture.
To enhance good loss landscape optimization, the AdamW optimizer [15] was used and configured with the following specifications:

- **Learning Rate (lr):** $1 \times 10^{-3}$
- **Weight Decay (wd):** $1 \times 10^{-2}$

This configuration was integrated with a learning rate scheduling strategy:

1. **Warm-up Phase:**
   - LinearLR scheduler with start_factor=0.1,
   - Duration: 5 epochs.
2. **Main Training Phase:**
   - CosineAnnealingWarmRestarts,
   - Initial period $\mathrm{T}_0 = 10$,
   - Period multiplication factor $\mathrm{T}_{\mathrm{mult}} = 2$.

The optimization process also incorporated:

- Label smoothing ($\epsilon = 0.1$) in the loss function,
- Gradient clipping with max_norm = 1.0,
- Mixed precision training using GradScaler.

The essence of these elaborate settings is to balance effective training dynamics with regularization, particularly considering our distributed training setup on the CIFAR-100 dataset.

# 3 Results and discussions

## 3.1 Experiment 1 results

The results obtained through the multi-GPU process using RAPIDS and DASK were very revealing. To begin with, the RAPIDS performance analysis is a great testament to how efficient and computationally fast distributed data-parallel training could be on multi-GPU system by leveraging RAPIDS' capabilities [22].

The performance of XGBoost for trip fare prediction (see Figs. 3, 4, 5) was superb. With a mean squared error of 10.87 and $R^2$ of 96.9%, the model demonstrated strong predictive performance.The MSE [9] is expressed as:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{9}$$

where

- MSE is the mean squared error and represents a measure of prediction accuracy,
- $n$ is the number of data points,
- $y_i$ is the actual value for the $i$-th data point,
- $\hat{y}_i$ is the predicted value for the $i$-th data point.

The $R^2$ [17] is expressed as:

$$R^2 = 1 - \frac{\text{SS}_{\text{res}}}{\text{SS}_{\text{tot}}}, \tag{10}$$

where

- $R^2$ is the coefficient of determination,
- $\text{SS}_{\text{res}}$ is the sum of squares of residuals,
- $\text{SS}_{\text{tot}}$ is the total sum of squares.

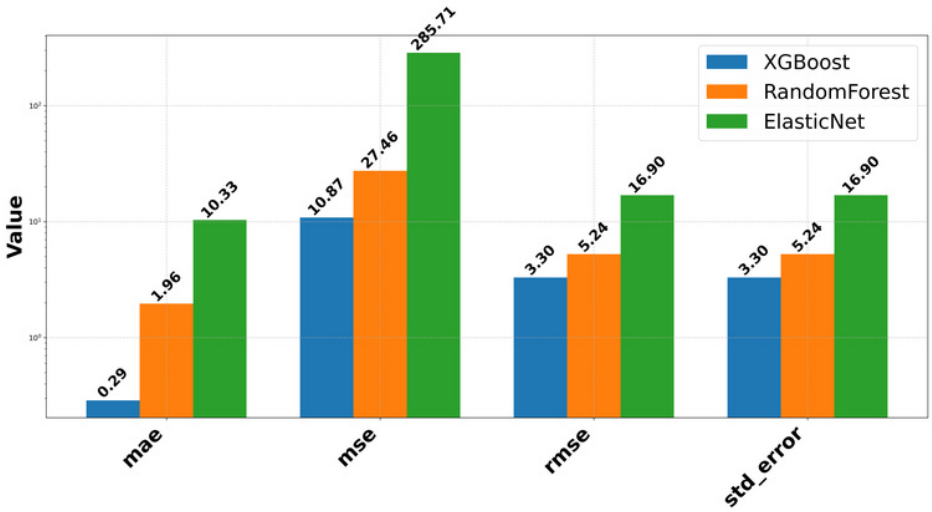



Figure 3: Error metrics comparison across models.

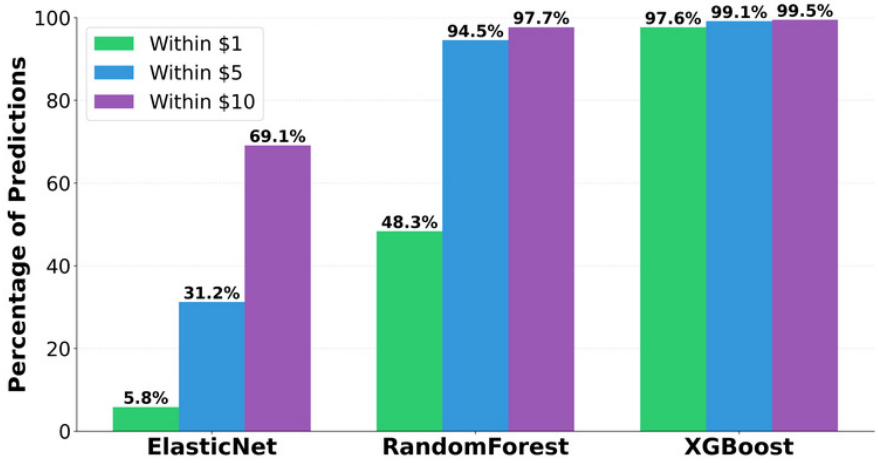



Figure 4: Fare predictions accuracy.

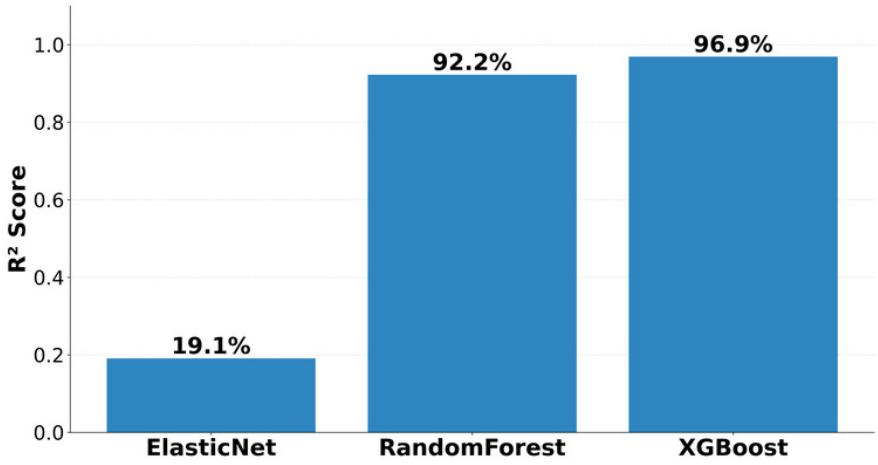
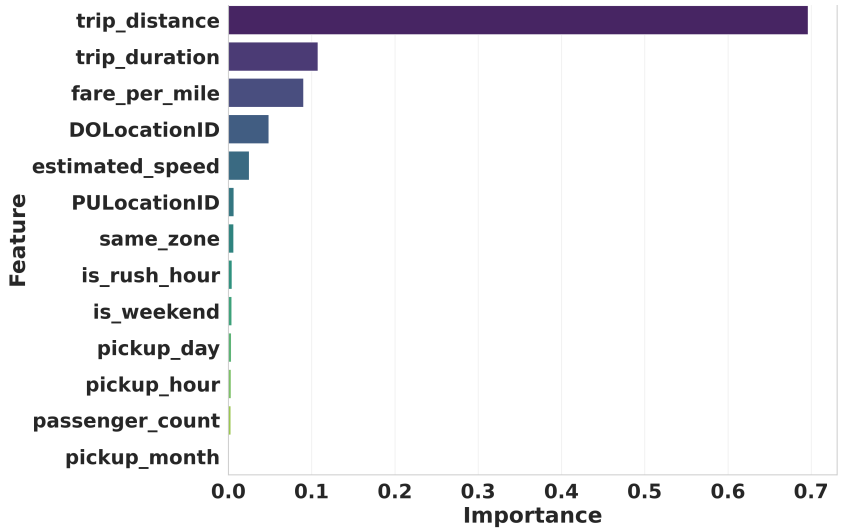



Figure 5: Training performance and model comparison.

The model's $R^2$ of 96.9% indicates that the model could explain nearly 97% of the variance in the taxi fare data. The high accuracy (ranging from 97.6% to 99.5%) within the $1 to $10 fare prediction range suggests its real-world applicability.

The top three features of importance (see Fig. 6) identified by the model were trip distance, trip duration, and fare per mile.




(a) Feature importance (XGBoost)

Figure 6: XGBoost feature importance.

Based on the high accuracy of the model, especially from the mean squared error (MSE), we analyzed Figs. 7 and 8 to develop the fare distribution patterns (see Table 1) and error distribution characteristics (see Table 2) respectively. From these analyses, we developed a **4Ps** model to illustrate its relevance. Each **P** is explained below.

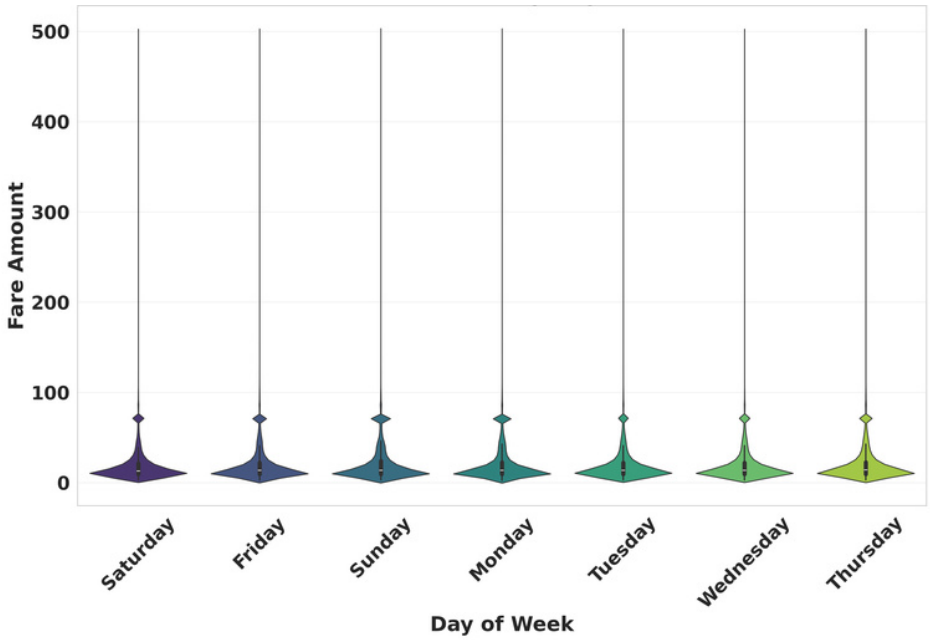



Figure 7: Error and fare distribution analysis.

| **Fare distribution feature** | **Description and analysis** |
|---|---|
| Median fare consistency | The median fare remains stable across weekdays, uninfluenced by erratic demand patterns at weekends and weekdays |
| High fare outliers | Long tails extending upwards indicate rare high fares, potentially related to long-distance or special cases |
| Symmetry across days | Fare distributions show consistent patterns, suggesting day-of-week factors minimally influence pricing |
| High-fare trips | Extended violin tops show rare high-fare trips (above $200), indicating optimization for shorter trips |

Table 1: Fare distribution analysis by day of the week.

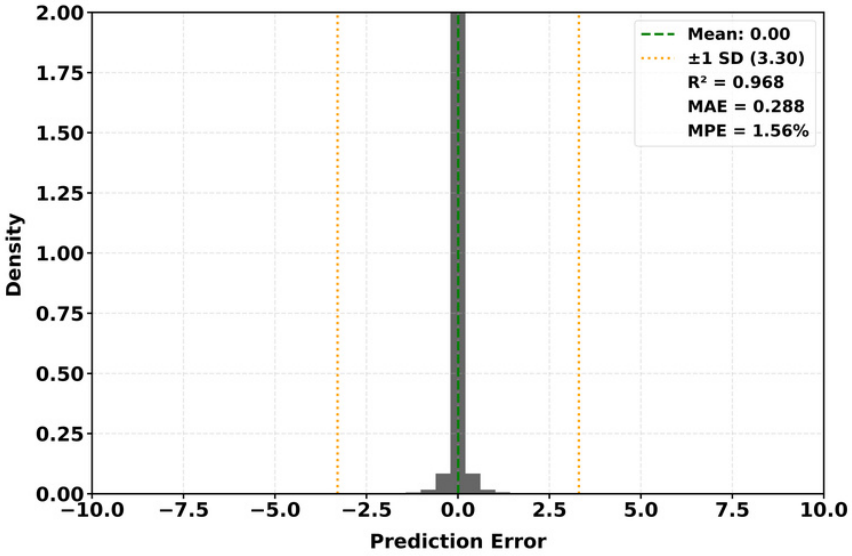



Figure 8: XGBoost error distribution.

**Planning:** The model revealed a strong correlation between trip distance, trip duration, and fare per mile. Transportation networks can utilize these features to forecast fares accurately, facilitating better scheduling and resource allocation.

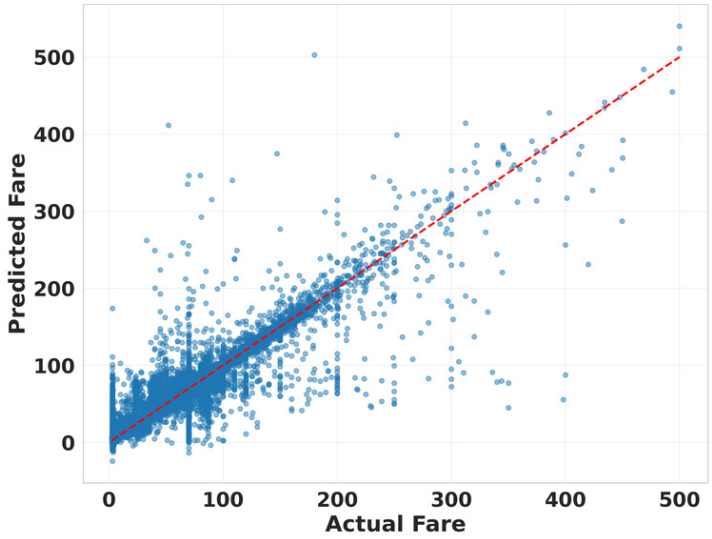
**Pricing:** The model's accuracy ensures fair pricing for both passengers and drivers. Passengers receive reliable fare estimates, while drivers can predict their earnings on a trip. With high predictive accuracy for lower fares, such services remain accessible to low-income earners. In conclusion, all stakeholders benefit from fair pricing.

**Prevention:** Transportation companies can use the model's accuracy to optimize fleet routes, helping to control fuel costs and improve service efficiency.

**Prediction:** In smart cities, this predictive model can aid traffic flow management and resource allocation, easing congestion issues.

| **Error distribution feature** | **Description and analysis** |
|---|---|
| Zero-centered distribution | Mean exactly at 0.00, indicating unbiased predictions with balanced errors |
| Error spread | Standard deviation of 3.30, with most errors falling within ±1 SD bounds |
| Model performance metrics | High $R^2$ (0.968) and low MAE (0.288) indicate excellent prediction accuracy |
| Error symmetry | Highly symmetric distribution around mean, with MPE of only 1.56% showing minimal systematic bias |

Table 2: Error distribution analysis of the prediction model.




(a) Predicted vs actual (XGBoost)

Figure 9: XGBoost model analysis.

These observations further confirm the model's reliability. The correlations make this model a valuable tool for transportation stakeholders in fare-related decision-making. Finally, the dense clustering of data points along the dotted diagonal line (see Fig. 9) demonstrates a strong linear correlation between predicted and actual fares.

The clustering also revealed near-perfect accuracy for fares in the $0 to $200 range, with increased scatter for fares above $200, indicating reduced accuracy for higher fares. The conspicuous lines visible in the NYC taxi fare prediction data represent systematic patterns validated through multiple forms of evidence. Our comprehensive analysis of the XGBoost model's predictions shows that these patterns are common to the taxi fare structure rather than spurious data points that the modeling system throws up. The first piece of evidence to support this is shown in Fig. 10.

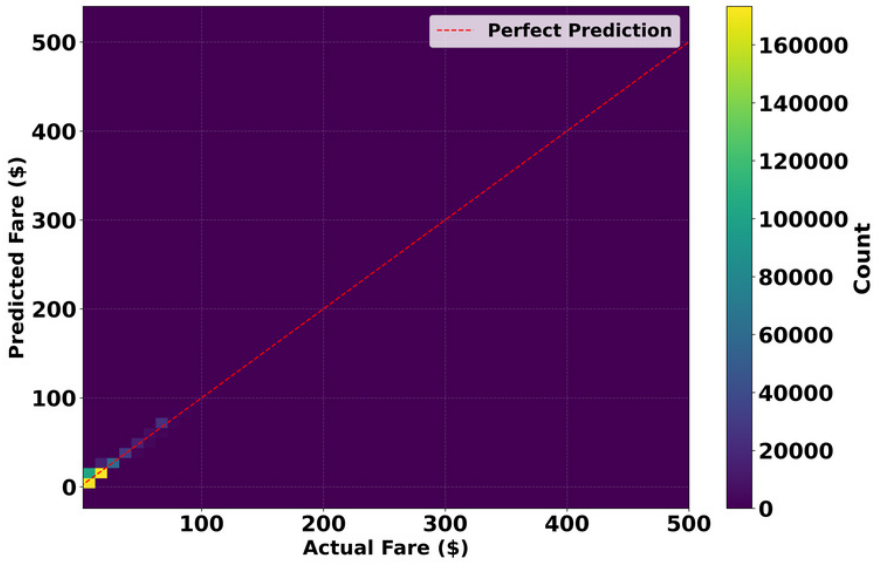



Figure 10: Fare prediction Density plot with systematic patterns in fare structure.

The density plot (see Fig. 10) shows clear clustering at regular intervals. This visual pattern is quantitatively supported by our statistical analysis, which shows remarkable prediction accuracy with 97.6% of predictions falling within $1 of actual fares and 99.1% within $5. The standard deviation of 3.30 further confirm the systematic nature of these patterns.

Furthermore, the second piece of evidence is revealed in Fig. 11 where 16 distinct high-frequency fare points that reinforce these patterns are clearly shown. The most common fare amount of $9.00 represents a typical short trip cost, while clear peaks appear at regulated price points like airport trips.

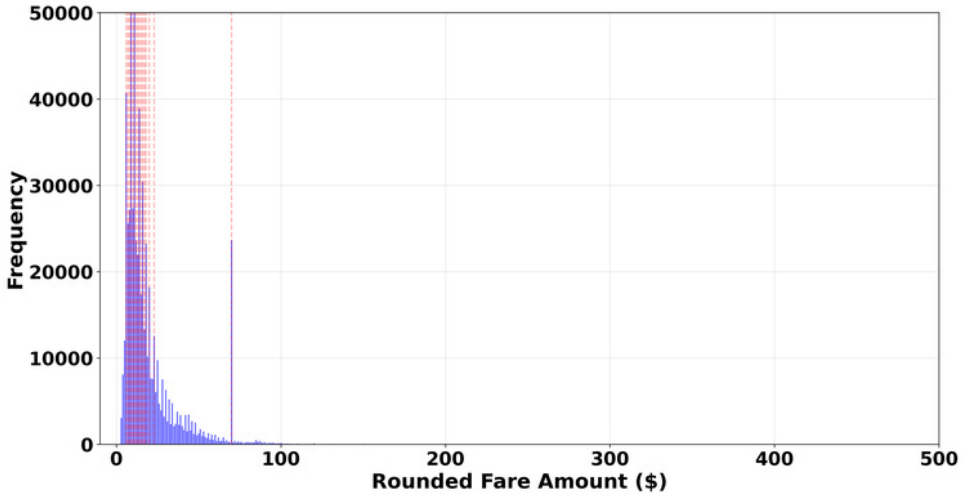



Figure 11: Fare amount Distribution with notable peaks.

This clustering directly reflects the NYC taxi fare structure, where prices are built from a $2.90 base fare with $0.50 increments per 1/5 mile or 60 seconds in traffic. The vertical lines in the data correspond to these standardized amounts, while horizontal striping emerges from the digital meter's rounding behavior.

Finally, the error distribution analysis (see Fig. 12) completes our analysis by showing how prediction accuracy varies across fare ranges. The pattern of increasing variance at higher fare percentiles aligns with the taxi industry's pricing structure, where longer trips accumulate more variable factors such as traffic conditions and route choices.

However, the consistent performance in common fare ranges (shown by tight error bounds in the middle percentiles) reveals that the model has correctly learned how the fare calculation works. These patterns emerge from the highly regulated nature of NYC taxi operations. Fixed-rate zones, especially for airport trips, create strong fare clusters. Standard route pricing and zone-based calculations produce regular fare intervals. Digital meter rounding to the nearest $0.50 generates the visible striping effect.

Together, these regulatory and operational factors explain why we observe lines at some specific standard price points.

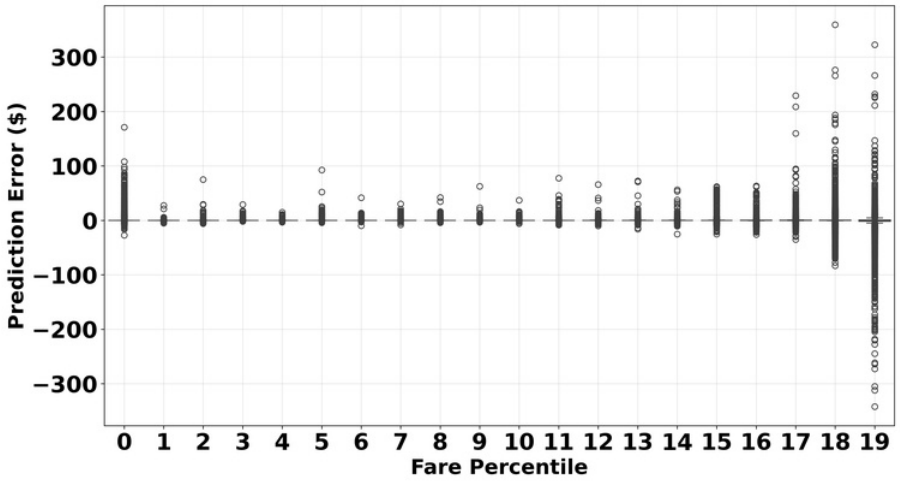



Figure 12: Error Distribution by Fare Range.

Figure 13 shows the training timing performances of the trained models. XGBoost took moderate time to complete the training and could still produce great predictive results. This is because of the efficient use of computer resources. Combining both factors of efficiency and accuracy makes it a good choice for real-world use since it can handle large taxi datasets without too much computer power. This efficiency aligns with sustainability goals by reducing energy use in computing, which helps create greener urban transport systems.

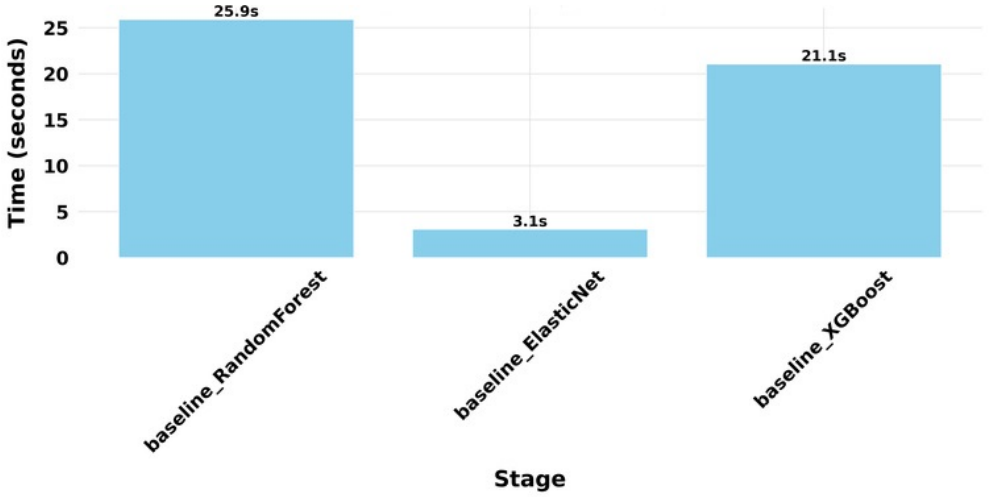



Figure 13: Baseline GPU training time.

## Scaling performance across GPUs

The varying speedup behavior shown by the traditional machine learning algorithms under consideration across multiple GPUs provides important insights into their practical deployment in distributed computing environments. We analysed the performances of the models using the metrics discussed below.

## Performance metrics

*Karp-Flatt metric*

Figure 14 shows the Karp-Flatt [13] metric analysis, which quantifies the serial fraction of the parallel implementation.

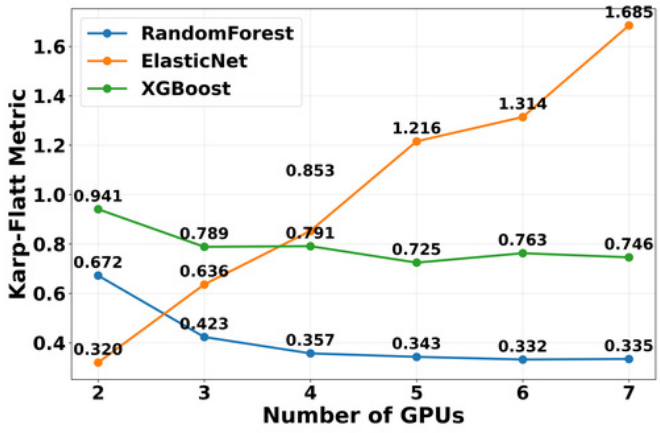



Figure 14: Karp-Flatt metric trends across GPU configurations.

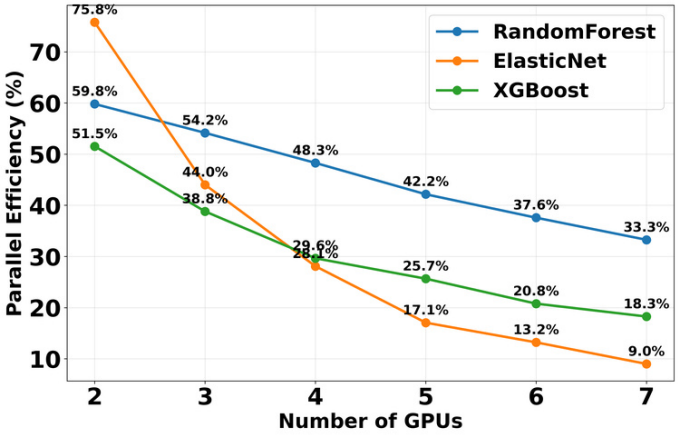



Figure 15: Parallel efficiency trends.

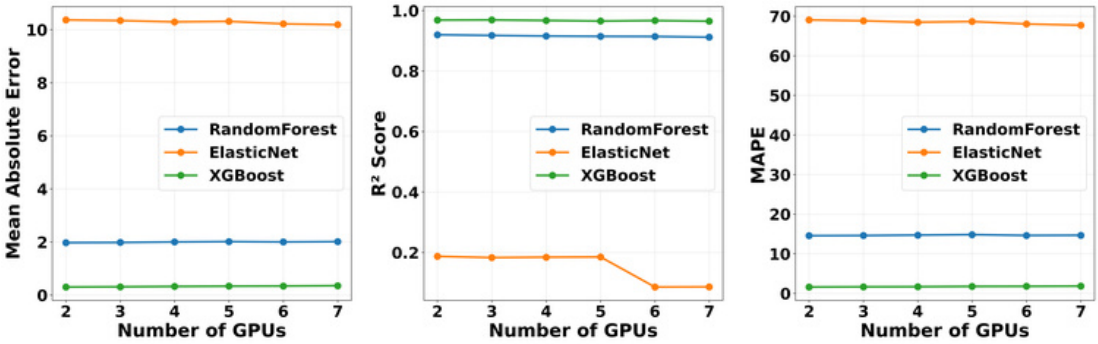



Figure 16: Performance metrics comparison across GPU configurations.

*Parallel efficiency*

The parallel efficiency results in Fig. 15 demonstrate the algorithms' ability to make use of additional computational resources effectively.

*Model performance indicators*

Figure 16 illustrates three key performance indicators across different GPU configurations.

*Speedup analysis*

The speedup characteristics, compared against ideal linear scaling, are presented in Fig. 17.

## Key observations

The analysis reveals several significant patterns:

- **RandomForest Performance:** Demonstrates the most favorable scaling characteristics with:

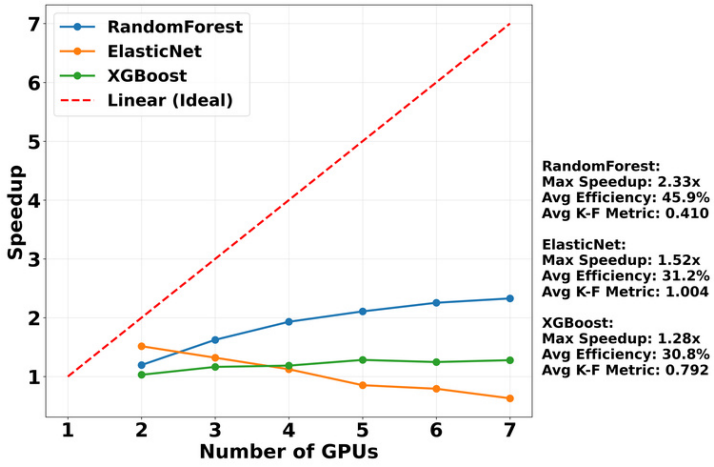



Figure 17: Speedup comparison against ideal linear scaling.

  - Maximum speedup of 2.33x,
  - Average efficiency of 45.9%,
  - Stable Karp-Flatt metric (0.410).

- **ElasticNet Behavior:** There is a notable performance decline as the GPUs were scaled:
  - Limited speedup (1.52x maximum),
  - Declining efficiency beyond 4 GPUs,
  - Highest Karp-Flatt metric (1.004), indicating significant serial fraction.
- **XGBoost Characteristics:** Exhibits moderate scaling properties:
  - Consistent but modest speedup (1.28x maximum),
  - Maintains stable efficiency (30.8% average),
  - Intermediate Karp-Flatt metric (0.792).

## Implications for deployment

These experimental findings have important implications for production deployment:

1. RandomForest presents the most cost-effective option for multi-GPU deployment, particularly in configurations up to 4 GPUs.
2. ElasticNet's performance suggests it is best suited for smaller GPU configurations ($\leq 4$ GPUs).
3. XGBoost offers a middle-ground solution with a predictable, scaling behavior.

The analyzed results reveal the essentiality of considering scaling characteristics when selecting algorithms for multi-GPU deployments, as the benefits of additional computational resources vary significantly among these popular machine learning approaches. For this setup, from all measures, it is very clear that ElasticNet is not an ideal model for use cases that are related to our setup.

## 3.2 Experiment 2 Results

For the CIFAR-100 dataset experiment, the distributed training on MobileNet-V3-Large showed promising results with very strong metrics that pointed to a very high computation efficiency.

### Performance analysis relative to sequential baseline

To understand our deep-learning model's performance in terms of multi-GPU parallelism, we carefully compare our distributed implementation against the sequential baseline following equation (6). As shown in Fig. 18, our sequential baseline ($T_1$) requires 52.24 seconds per epoch, while the parallel implementation ($T_p$) achieves 8.03 seconds per epoch using 7 GPUs.

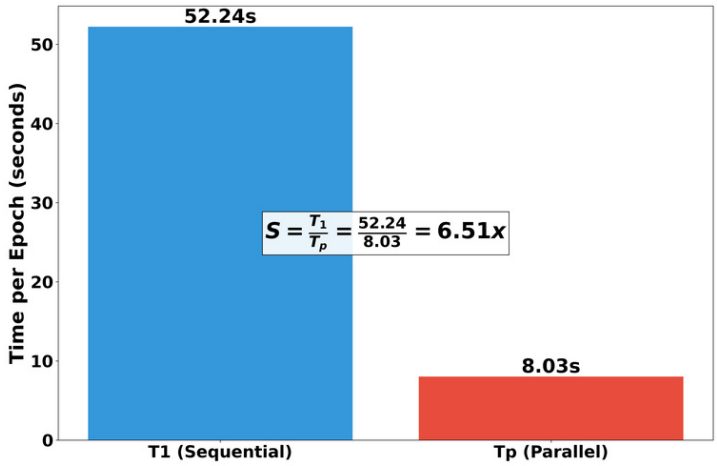



Figure 18: Comparison of sequential baseline ($T_1$) versus parallel implementation ($T_p$).

This results in a speedup of 6.51x, demonstrating strong scaling efficiency. To further analyze the scaling behavior, Fig. 19 compares our actual speedup against ideal linear scaling.

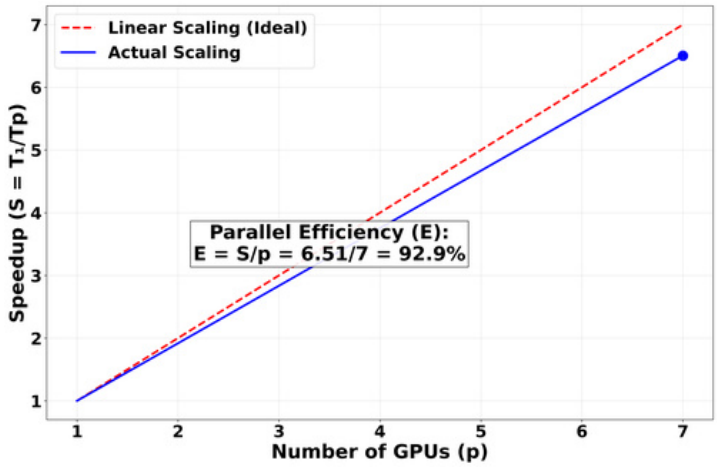



Figure 19: Scaling analysis showing actual speedup versus ideal linear scaling.

The parallel efficiency, calculated as $E = S/p = 6.51/7 = 92.9\%$, demonstrates that our implementation maintains excellent scaling properties, achieving near-linear speedup.

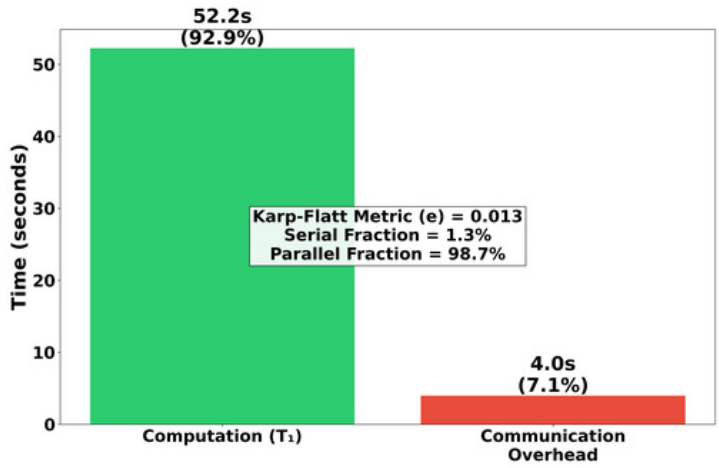



Figure 20: Time breakdown between computation and communication overhead.

To quantify the parallelization overhead, we employ the Karp-Flatt metric:

$$e = \frac{1/S - 1/p}{1 - 1/p}. \tag{11}$$

As shown in Fig. 20, our implementation achieves:

- A low Karp-Flatt metric of 0.013, indicating minimal serialization,
- 98.7% parallel fraction, demonstrating excellent parallelization,
- Only 7.1% communication overhead relative to computation time.

These metrics comprehensively demonstrate that our distributed implementation achieves:

1. Strong speedup (6.51x) relative to the sequential baseline.
2. Near-linear scaling efficiency 92.9%.
3. Minimal parallel overhead (Karp-Flatt = 0.013).

The performance results show that our implementation effectively utilizes the available GPU resources while maintaining minimal communication overhead, as evidenced by both the high parallel efficiency and low Karp-Flatt metric.

## The model performance metrics

The receiver operating curves (ROC) in Fig. 21 reveal another vivid information about this distributed data-parallel (DPP) training model. The area under the curve (ROC-AUC) for the model showed high predictive power, with a higher number of counts clustering around 0.9 to close to 1.0. This means that the model classifies well above the random classification.

Key findings from the ROC curves:

- A significant portion of classes achieved high ROC-AUC scores above 0.95

- Most classes clustered in the high-performance range of 0.9-1.0
- Very few classes showed poor performance (below 0.8)
- The distribution shows consistent performance across different classes

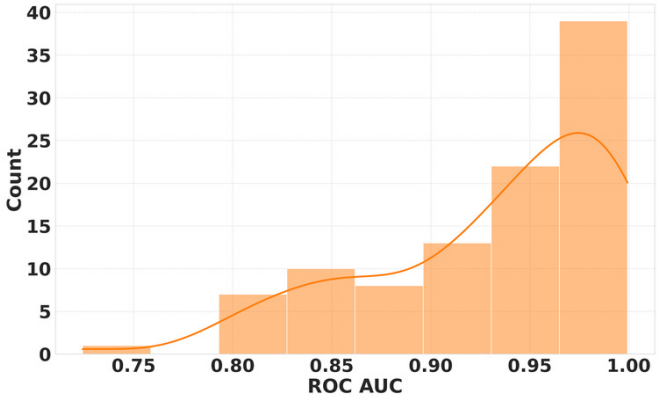



Figure 21: ROC curves visualization.

From Fig. 22, the radar plot gave significant insights into the accuracy level of the top 10 classes, with classes 68 and 53 gravitating greatly towards the 90% accuracy point.

Analysis of top-performing classes:

- Classes 68 and 53 achieved the highest accuracy (near 90%)
- Top 10 classes showed consistent high performance
- Even distribution of learning across the major classes
- The high performance can be attributed to:
  - Well-balanced training data
  - Clear distinguishing features in these classes
  - Effective distributed training approach
  - Sufficient data representation for each class

The model's computational performance and efficiency were relatively stable throughout the training epochs as could be deduced from the execution time (see Fig. 23) that stayed within the range of 9.5 to 12.5 seconds. The dips and spikes notwithstanding, the GPU utilization efficiency was still relatively stable. The periodic spikes in execution time could be attributed to some chosen implementation factors associated with the experiment. These factors are succinctly explained below:

1. The primary contributor is our learning rate scheduling implementation using CosineAnnealingWarmRestarts ($T_0$=10, $T_{mult}$=2).
   These arrangements create predictable periodic warm restarts at exponentially increasing intervals of 10, 20, 40, ...70).

2. These spikes are amplified by the synchronization overhead from our DistributedData-Parallel (DDP) training setup, particularly during learning rate reset points.

3. Additional factors include mixed-precision training operations (GradScaler updates) and CUDA memory management events, which tend to align with these restart points.

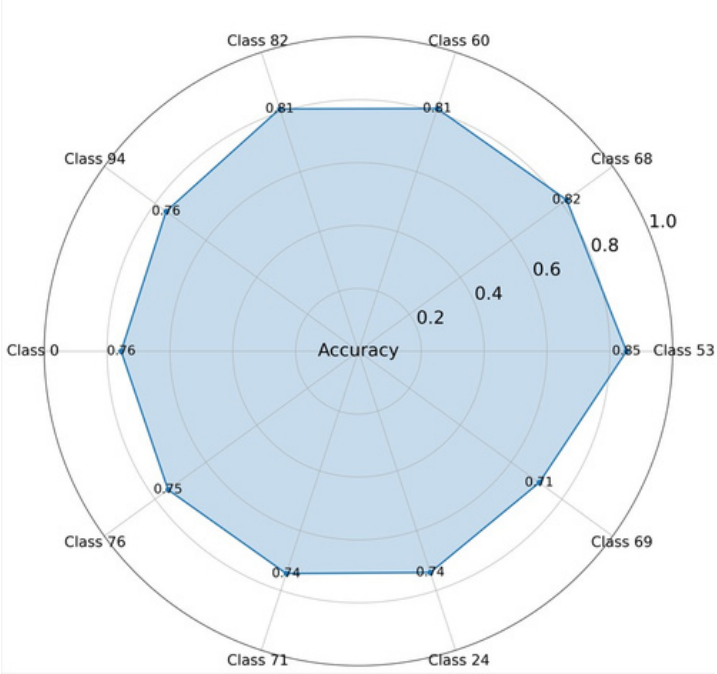



Figure 22: Radar plot for top 10 classes.

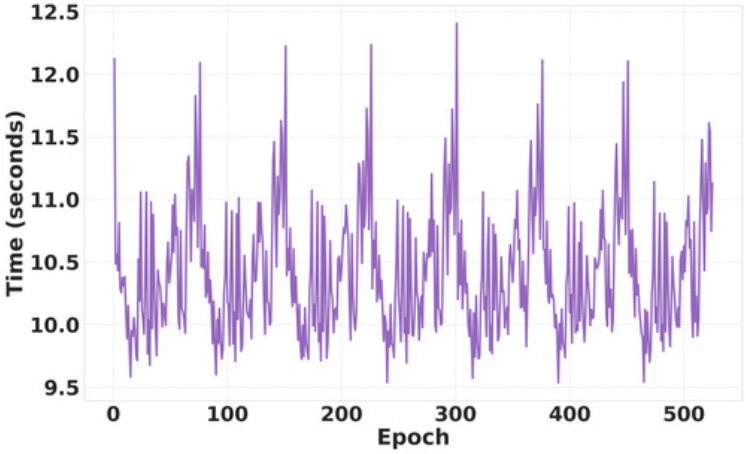



Figure 23: Total execution time per epoch.

While beneficial for optimization, these architectural design choices create the observed periodic pattern in execution time. The regularity of these spikes corresponds to the deterministic nature of our warm restart schedule, while their varying intensities reflect the combined effect of these factors at restart points.

We analyzed the training and validation loss distributions shown in Figs. 24 and 25, with the key statistics summarized in Table 3. The table provides a detailed analysis of the model's performance through loss ranges, distribution shapes, and comparative patterns between training and validation losses. The training and validation losses (see Fig. 25) exhibited variance across the training epochs.


Figure 24: Training and Validation losses distribution.

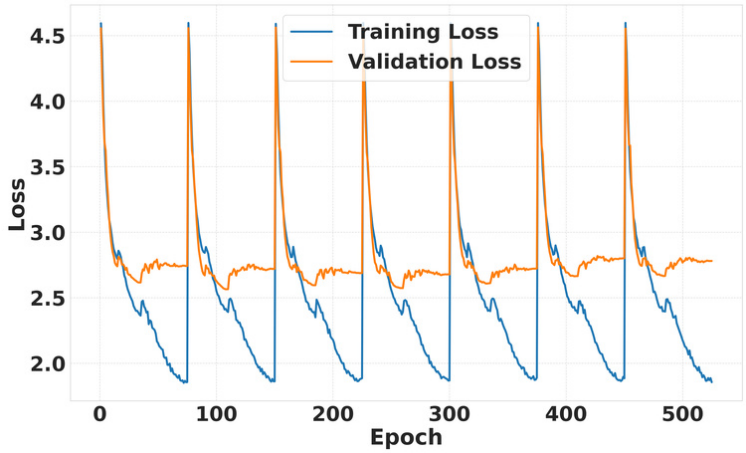



Figure 25: Training and validation loss.

While the training losses dipped toward the minimum, the same was not true for validation losses as they played more in the regions between 2.7 and 3.0.

| Metric | Training loss | Validation loss |
|---|---|---|
| Range | ∼1.5 to 4.8 | ∼2.5 to 4.7 |
| Center | Most values centered around 2.0 to 3.0 | Most values centered around 2.7 to 3.0 |
| Shape | Wide distribution with significant fluctuations and outliers | Narrower distribution with tighter variance |
| Implication | Indicates the model is still learning but may be overfitting in some cases | Consistent performance on validation data, suggesting good generalization |
| Comparison | Wider loss distribution than validation, possible overfitting or instability | Indicates better generalization but refinement needed for training loss |

Table 3: Comparison of training and validation loss distribution.

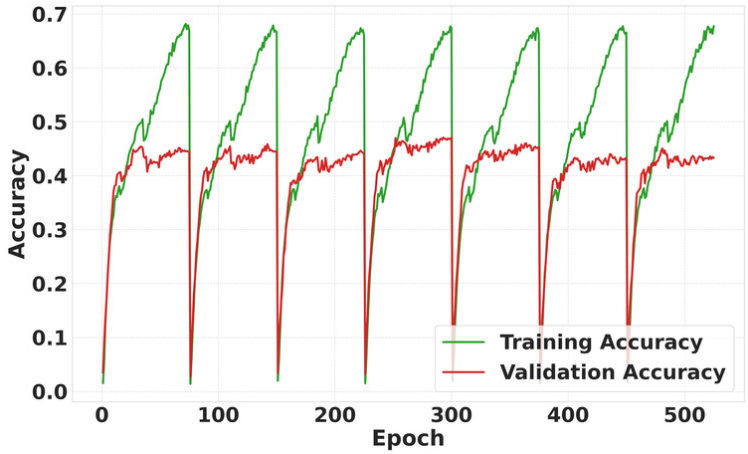



Figure 26: Training and validation accuracy.

The loss performances are ultimately reflected in the model's accuracy as shown in Fig. 26, where there were many discrepancies between the training and validation accuracies across epochs. The training accuracy (green) consistently climbs to peaks around 0.65-0.7, while the validation accuracy (red) plateaus at approximately 0.45. This significant gap between training and validation accuracy suggests overfitting, where the model performs well on training data but fails to generalize effectively to new data. Despite the hyperparameter tuning that is systematically incorporated into the training process, the validation accuracy remains relatively stable at around 0.45, indicating that the model's ability to generalize isn't improving with additional training epochs.

Finally, the model performed well on the top 20 classes (see Fig. 27 ) with predicted labels and true labels being accurately the same in most cases as shown by the confusion matrix [21] which is mathematically represented as:

$$C_{ij} = \sum_{k=1}^{n} \mathbf{1}(y_k = i \wedge \hat{y}_k = j), \tag{12}$$

where

- $C_{ij}$ is the confusion matrix entry for true label $i$ and predicted label $j$,
- $y_k$ is the true label for the $k$-th sample,
- $\hat{y}_k$ is the predicted label for the $k$-th sample,
- $\mathbf{1}(\cdot)$ is the indicator function.

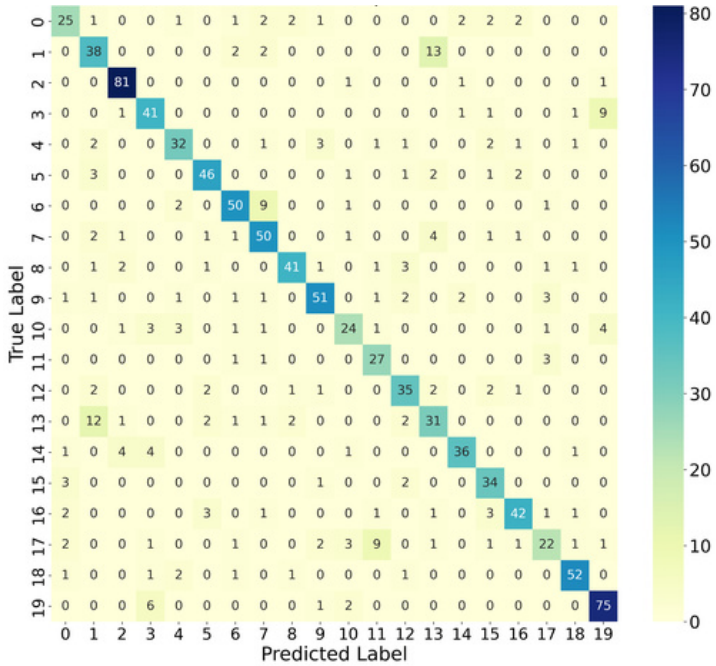



Figure 27: Confusion matrix for top 20 classes.

# 4 Conclusion and future work

This work has revealed a great deal of information relating to parallel programming performances in relation to efficiency, scalability, speedup and adaptability, while using RAPIDS libraries across multiple GPUs. The first experiment has shown that with proper experimental setup and relevant model pipelines, parallel programming could be used to design scientific models that would be robust enough to have application in the real-world dataset, and as such, informative and impactful details could be seen promptly from data points which might be very difficult to achieve when working on only CPU-based machine. This was aptly demonstrated with the large parquet formatted dataset [1] that was used for the trip fare prediction. We also

showed the possible trade-offs that could be considered when there is a need to use any of the analyzed models.

The same is equally true for the complex CIFAR-100 dataset that has applications in real-world scenarios in areas like object recognition, vision systems, image detection, mobile applications, and more. The parallelized model could be embedded in real-life gadgets for the use of humanity as the experiment has shown its excellent performance in the analyzed use case in relation to the scaling metrics' results, where the **parallel efficiency** $\eta$ and **speedup factor** $S(n)$ achieved align well with theoretical predictions. The combination of RAPIDS, DASK, and PyTorch provided a powerful framework for handling large-scale datasets and complex deep learning models.

In our future work, we would explore how some complex datasets could be finely trained on the parallel system using deep learning architecture with a control handle on overfitting. Furthermore, to eradicate some inefficiencies in the trio setup, further work would be done on the optimized implementation of RAPIDS setup and DASK without PyTorch. This way, architectural complexities would be simplified, as the design would focus on the duo of RAPIDS and DASK frameworks. Additionally, the conflict that could arise from the framework's memory management system which may result in fragmentation and overhead would be minimized. On a final note, the constant data format conversions between PyTorch tensors and RAPIDS/DASK data structures with the associated unnecessary bottlenecks in the processing pipeline would be eradicated.

## Acknowledgements

The research leading to these results has received funding from the Instituto Politécnico Nacional (IPN), under research projects SIP20240164 and SIP20241558, and the Consejo Nacional de Humanidades Ciencias y Tecnologias (CONAHCYT) through project CF-2023-I-108.

Taurean Dyer(tdyer@nvidia.com), Senior Product Manager/Technical Product Manager for Data Science and AI Infrastructure at NVIDIA, provided RAPIDS tutorials.

## References

[1] D. Abadi, R. Agrawal, A. Ailamaki, M. Balazinska, P. A. Bernstein, M. J. Carey, S. Chaudhuri, J. Dean, A. Doan, M. J. Franklin, et al. Parquet: Efficient data storage for big data applications. In *Proceedings of the 2024 ACM SIGMOD International Conference on Management of Data*, pages 1631–1636. ACM, 2024.

[2] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS spring joint computer conference*, pages 483–485, 1967.

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[4] R. E. Bryant and D. O'Hallaron. *Computer Systems: A Programmer's Perspective*. Pearson Education, 3rd edition, 2016.

[5] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[6] N. Ferreira, J. Poco, H. T. Vo, J. Freire, and C. T. Silva. Visual exploration of big spatio-temporal urban data: A study of New York city taxi trips. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2149–2158, 2013.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[8] J. L. Gustafson. Reevaluating Amdahl's law. *Communications of the ACM*, 31(5):532–533, 1988.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[10] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Elsevier, 2011.

[11] A. Howard, M. Tan, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1314–1324, 2023.

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. In *arXiv preprint arXiv:1704.04861*, 2017.

[13] A. H. Karp and H. P. Flatt. Measuring parallel processor performance. *Communications of the ACM*, 33(5):539–543, 1990.

[14] X. Li, G. Pan, Z. Lei, and Z. Huang. Spatiotemporal-aware neural networks for taxi demand prediction. *IEEE Transactions on Intelligent Transportation Systems*, 24(5):4912–4923, 2023.

[15] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2023.

[16] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012.

[17] N. J. D. Nagelkerke. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3):691–692, 1991.

[18] New York City Taxi and Limousine Commission. NYC TLC trip record data, 2024. Accessed: October 2024.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, and A. Desmaison. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 36, pages 8026–8037, 2023.

[20] M. Poongodi, M. Malviya, C. Kumar, M. Hamdi, V. Vijayakumar, J. Nebhen, and H. Alyamani. New York city taxi trip duration prediction using MLP and XGBoost. *International Journal of System Assurance Engineering and Management*, pages 1–12, 2022.

[21] D. M. Powers. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[22] RAPIDS AI. RAPIDS: Collection of libraries for end-to-end GPU data science, 2023. Accessed: September 2024.

[23] S. Raschka, J. Patterson, and C. Nolet. Machine learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence. *Information*, 11(4):193, 2022.

[24] S. Wang, H. Zheng, X. Wen, and S. Fu. Distributed high-performance computing methods for accelerating deep learning training. *Journal of Knowledge Learning and Science Technology*, 3(3):108–126, 2024.

[25] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.