CS6847
CLOUD COMPUTING

ASSIGNMENT 3

Name: **Rudra Pratap Singh**
Roll Number: **EE22B171**

1 Dataset Description

The dataset used in this project is derived from the NYC Taxi Fare Prediction task. It contains trip-level information describing individual taxi rides, including the start time, pickup and dropoff coordinates, and number of passengers. The training set includes the target variable *fare_amount*, while the test set omits this field since it must be predicted.
The features include: `pickup_datetime`, `pickup_longitude`, `pickup_latitude`, `dropoff_longitude`, `dropoff_latitude`, and `passenger_count`. The target is `fare_amount` (available in the training set). A total of 500,000 training samples and 9,914 test samples were used. Preprocessing was minimal and consistent across train/test.

1.1 Input Features

The primary features contained in both train and test CSV files are:

- **pickup_datetime** — Timestamp indicating when the ride began.
- **pickup_longitude, pickup_latitude** — GPS coordinates at trip origin.
- **dropoff_longitude, dropoff_latitude** — GPS coordinates at trip destination.
- **passenger_count** — Number of passengers in the vehicle.

The target feature present only in the training data is:

- **fare_amount** — The dollar amount (USD) charged for the ride.

The raw dataset is approximately 5 GB in size and was streamed in chunks to avoid memory overflow in Colab environments.

2 Preprocessing

A lightweight, deterministic preprocessing pipeline was applied uniformly to both the training and test sets to ensure consistent feature shapes and to avoid dropping rows. The pipeline implementation (function `preprocess_df`) performed the following steps:

1. If the column `fare_amount` is not present (test set), a placeholder column of zeros is added so the pipeline always returns a target vector with the same shape.
2. Convert `pickup_datetime` to a pandas datetime and extract temporal features:
   - `hour` (hour of day),
   - `dow` (day of week),
   - `month`.
3. Convert numeric columns using `pd.to_numeric(..., errors='coerce')` for: `pickup_longitude, pickup_latitude, dropoff_longitude, passenger_count, fare_amount,dropoff_latitude`.
4. Missing numeric values are filled with zeros (explicit `.fillna(0)`), so no rows are removed by the pipeline.

5. Compute geodesic distance (Haversine) between pickup and dropoff points. In this implementation:

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cos\phi_2 \sin^2\left(\frac{\Delta\lambda}{2}\right), \qquad d = 2R \operatorname{atan2}\left(\sqrt{a}, \sqrt{1-a}\right),$$

where ϕ are latitudes (radians), λ are longitudes (radians), $\Delta\phi = \phi_2 - \phi_1$, $\Delta\lambda = \lambda_2 - \lambda_1$, and $R = 6371$ km.

6. Construct additional features:
   - `abs_lon_diff` = |pickup_longitude - dropoff_longitude|,
   - `abs_lat_diff` = |pickup_latitude - dropoff_latitude|.
7. Cast `passenger_count` to integer and clip values to the range $[0, 10]$.

The function returns feature matrix X and target vector y. The exact feature set returned (and used for training and inference) is:

{`hour, dow, month, pickup_longitude, pickup_latitude, dropoff_longitude, dropoff_latitude, distance_km, abs_lon_diff, abs_lat_diff, passenger_count`}.

Notes: the pipeline intentionally does not drop rows or apply outlier caps; numeric missingness is handled by zero-imputation and a `fare_amount` placeholder is added for test data so downstream scaling and model code can operate without conditional branches.

3 Neural Network Architecture

Three neural network sizes were implemented. The final architectures (as compiled and trained on CPU/GPU/TPU) are:

3.1 Small Network

- 2 hidden layers
- 64 and 32 units respectively
- ReLU activation

3.2 Medium Network

- 3 hidden layers
- 128, 128, and 64 units
- ReLU activation

3.3 Large Network

- 4 hidden layers
- 256, 128, 64, and 32 units
- ReLU activation

All networks used:

- Mean Squared Error (MSE) loss
- Adam optimizer
- 25 epochs (with early stopping allowed)
- Normalized inputs using StandardScaler

Table 1: Input–Output Dimensions of Each Layer for All Neural Network Models

Model	Layer	Input Shape	Output Shape
Small	InputLayer	–	(None, 11)
	Dense 1	(None, 11)	(None, 64)
	Dense 2	(None, 64)	(None, 32)
	Output Dense	(None, 32)	(None, 1)
Medium	InputLayer	–	(None, 11)
	Dense 1	(None, 11)	(None, 128)
	Dense 2	(None, 128)	(None, 128)
	Dense 3	(None, 128)	(None, 64)
	Output Dense	(None, 64)	(None, 1)
Large	InputLayer	–	(None, 11)
	Dense 1	(None, 11)	(None, 256)
	Dense 2	(None, 256)	(None, 128)
	Dense 3	(None, 128)	(None, 64)
	Dense 4	(None, 64)	(None, 32)
	Output Dense	(None, 32)	(None, 1)

4 Training Configuration

Experiments were executed separately on:

- Google Colab CPU runtime,
- Google Colab GPU runtime,
- Google Colab TPU v2 runtime.

The same model code, hyperparameters, and preprocessing were used for all devices. Results (RMSE, MAE, R^2, epoch time, total time) were saved into Google Drive for reproducibility.

5 Execution Time Comparison

Figure 1 summarizes the average epoch time measured for each device:

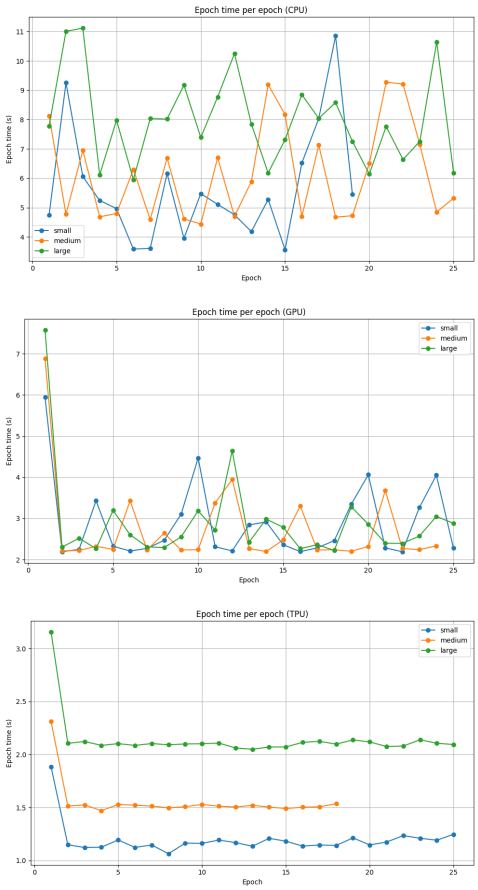


Figure 1: Epoch time comparison across CPU, GPU, and TPU.

5.1 Epoch Time Comparison Across CPU, GPU and TPU

Figures 1 show the per-epoch training time for the small, medium, and large neural network architectures on CPU, GPU, and TPU respectively.

On the **CPU**, the epoch times fluctuate between 4–11 seconds depending on model size. The large model consistently exhibits the highest per-epoch cost, while the small model is the fastest. The variability is also higher on CPU, indicating sensitivity to background system load and thread scheduling.

On the **GPU**, all three network sizes train significantly faster, with epoch times mostly in the range of 2–4 seconds. The GPU provides a clear and stable acceleration compared to the CPU, especially for the medium and large models where parallel computation is exploited effectively.

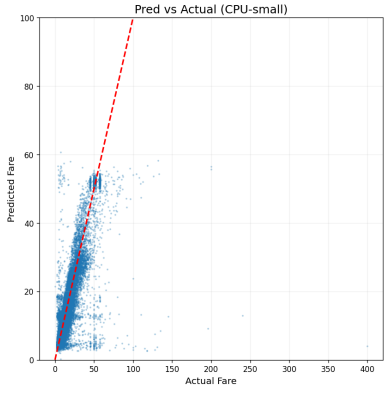
On the **TPU**, the per-epoch time is the lowest among all devices. The small model consistently trains in approximately 1.1–1.3 seconds per epoch, while the medium and large models remain below 2.2 seconds on average. The plots also show that TPU training is more stable, with minimal per-epoch fluctuation.

Overall, the results confirm that:

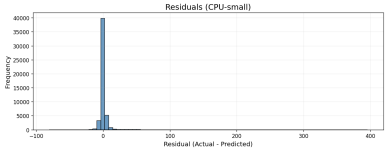
- GPU is considerably faster than CPU for all model sizes.

- TPU provides the highest and most stable training throughput.
- Larger networks scale efficiently on GPU and TPU but suffer higher overhead on CPU.

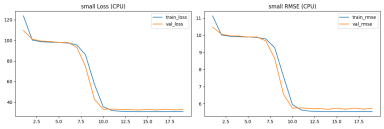
5.2 CPU Figures



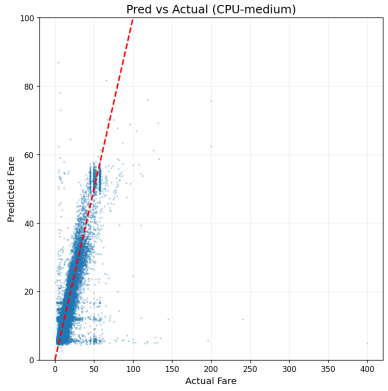
(a) Pred vs Actual (CPU - Small)



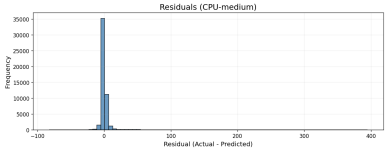
(b) Residuals (CPU - Small)



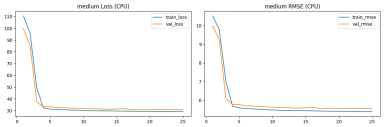
(c) Loss / RMSE (CPU - Small)



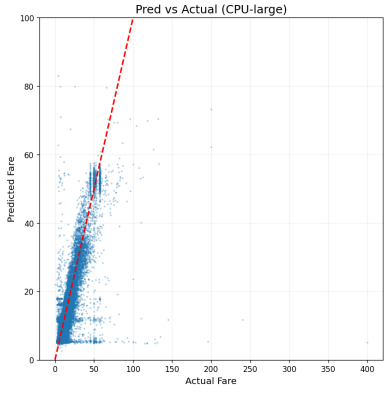
(d) Pred vs Actual (CPU - Medium)



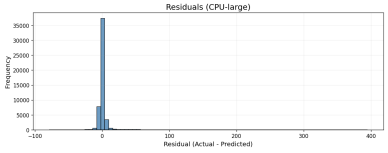
(e) Residuals (CPU - Medium)



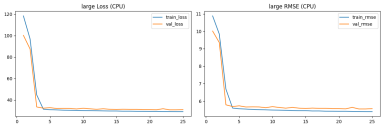
(f) Loss / RMSE (CPU - Medium)



(g) Pred vs Actual (CPU - Large)



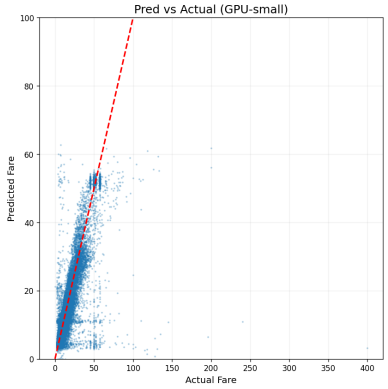
(h) Residuals (CPU - Large)



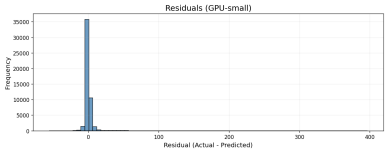
(i) Loss / RMSE (CPU - Large)

Figure 2: CPU diagnostics: predicted vs actual, residuals and training loss curves.

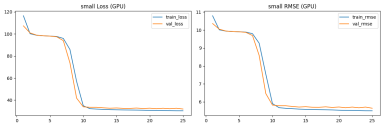
5.3 GPU Figures



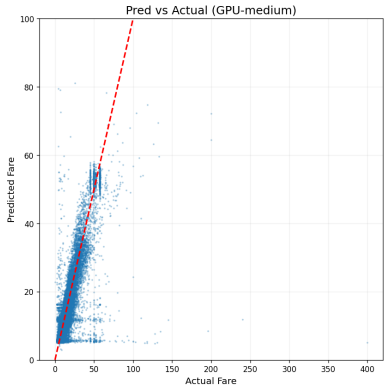
(a) Pred vs Actual (GPU - Small)



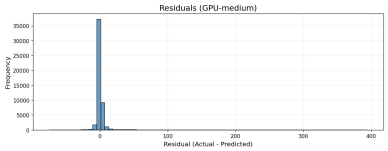
(b) Residuals (GPU - Small)



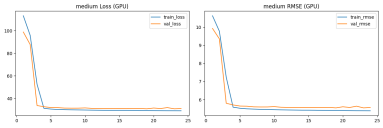
(c) Loss / RMSE (GPU - Small)



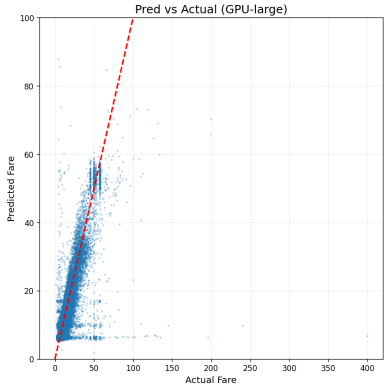
(d) Pred vs Actual (GPU - Medium)



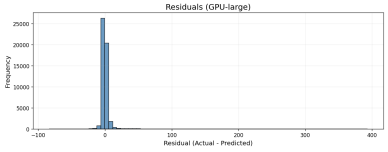
(e) Residuals (GPU - Medium)



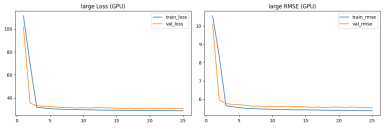
(f) Loss / RMSE (GPU - Medium)



(g) Pred vs Actual (GPU - Large)



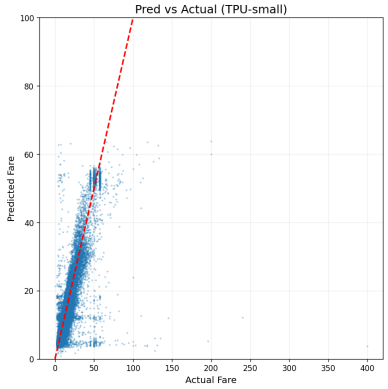
(h) Residuals (GPU - Large)



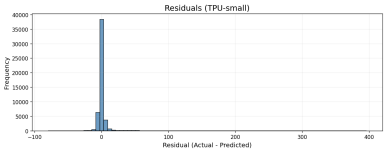
(i) Loss / RMSE (GPU - Large)

Figure 3: GPU diagnostics: predicted vs actual, residuals and training loss curves.

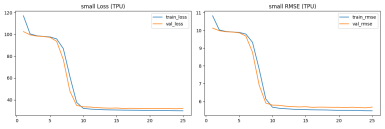
5.4 TPU Figures



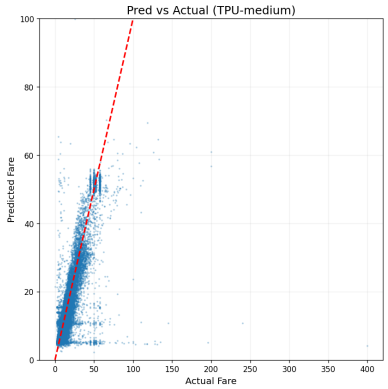
(a) Pred vs Actual (TPU - Small)



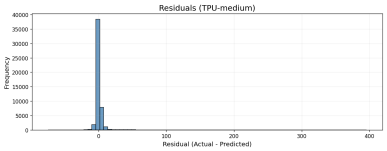
(b) Residuals (TPU - Small)



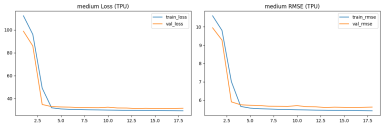
(c) Loss / RMSE (TPU - Small)



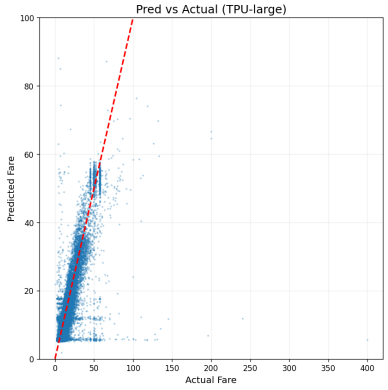
(d) Pred vs Actual (TPU - Medium)



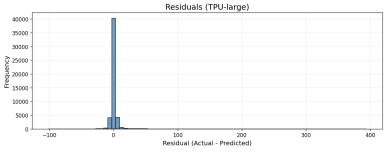
(e) Residuals (TPU - Medium)



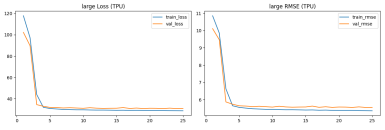
(f) Loss / RMSE (TPU - Medium)



(g) Pred vs Actual (TPU - Large)



(h) Residuals (TPU - Large)



(i) Loss / RMSE (TPU - Large)

Figure 4: TPU diagnostics: predicted vs actual, residuals and training loss curves.

6 Results

6.1 Summary Tables

Table 2: CPU: Training performance and validation metrics

Size	Epochs	Avg Epoch Time (s)	Total Train Time (s)	RMSE	MAE	R^2
Small	19	5.621	107.104	5.680	2.501	0.672
Medium	25	6.166	154.402	5.546	2.459	0.687
Large	25	8.011	200.518	5.548	2.385	0.687

Table 3: GPU: Training performance and validation metrics

Size	Epochs	Avg Epoch Time (s)	Total Train Time (s)	RMSE	MAE	R^2
Small	25	2.869	71.989	5.645	2.479	0.676
Medium	24	2.740	65.989	5.543	2.480	0.688
Large	25	2.905	72.892	5.538	2.430	0.688

Table 4: TPU: Training performance and validation metrics

Size	Epochs	Avg Epoch Time (s)	Total Train Time (s)	RMSE	MAE	R^2
Small	25	1.195	29.982	5.637	2.502	0.677
Medium	18	1.555	28.084	5.598	2.429	0.682
Large	25	2.140	53.612	5.554	2.406	0.686

7 Conclusion

This project evaluated the training performance and predictive accuracy of three neural network architectures (small, medium, large) across three computational platforms: CPU, GPU, and TPU. The results consistently demonstrate that hardware choice has a significant impact on training efficiency, while predictive accuracy remains relatively stable across devices.

The CPU exhibited the slowest training times, with higher per-epoch variability and noticeably longer total training durations. In contrast, the GPU provided a substantial acceleration, reducing epoch times by more than half for all model sizes. The TPU achieved the best overall performance, delivering the lowest and most stable epoch durations and completing training with the shortest total time.

Despite large differences in training speed, validation accuracy remained comparable across all devices. The medium and large models on GPU and TPU achieved the strongest predictive performance, reflected by lower RMSE and MAE values and higher R^2 scores. Residual and scatter-plot analyses further confirmed that predictions were generally stable, with a slight tendency to underestimate very high fares.

Overall, the TPU large model achieved the most favorable balance between training speed and predictive accuracy. The GPU medium model offered the most consistent validation performance with efficient training time. Across all experiments, the results clearly highlight the advantages of specialized accelerators (GPU and TPU) for deep learning workloads, especially when training on large real-world datasets.