Parallel and Distributed Computing

Assignment 2

Data and Task Parallelism for ML Applications

Submitted by:
Syed Abdullah Shah
Ahsan Rasheed
Ebad Afridi
Talha Abdulrehman
Faris Ahmad Nasir
Muhammad Owais

Submitted to:
Mr. Qazi Haseeb Yousaf

Date: 3rd October, 2025

Contents

Acknowledgement

We would like to acknowledge the use of Artificial Intelligence (AI) tools as a supportive resource in preparing this assignment. AI was used as a helping tool for structuring the report, generating ideas, and improving clarity. However, the implementation, analysis, and results are based on our own efforts and understanding of the course material.

1 Introduction

The rapid growth of machine learning (ML) applications has made it necessary to explore efficient computational strategies for training and inference. Traditional sequential execution on CPUs often becomes a bottleneck, especially when working with large datasets such as credit card transactions, movie reviews, or real-time social media feeds. Parallel and distributed computing provides solutions to these challenges by leveraging modern hardware architectures. Two key strategies are:

- **Data Parallelism:** Distributing large datasets across multiple processing units (e.g., GPU cores) to perform computations simultaneously. This is highly beneficial for deep learning models where large batches of data must be processed.
- **Task Parallelism:** Executing multiple independent or semi-independent tasks concurrently using CPU multithreading. This is effective when tasks can run in parallel, such as training separate models or splitting preprocessing workloads.

In this assignment, we apply these parallel computing strategies to three machine learning problems:

1. **Credit Card Fraud Detection** – a binary classification task.
2. **IMDB Sentiment Analysis** – a natural language processing (NLP) task classifying reviews as positive or negative.
3. **Twitter Sentiment Analysis** – a text classification task involving short, noisy social media posts.

By comparing GPU-based data parallelism and CPU-based task parallelism, the report aims to highlight their effectiveness, runtime differences, and scalability.

2 Problem Statement

Machine learning algorithms often demand significant computational resources to process high-dimensional data and train deep models. Running these models on a single CPU core leads to inefficiency and long execution times.

The primary challenges addressed in this assignment are:

- **High Computational Cost:** Large datasets such as the IMDB reviews (50,000 samples) or credit card fraud detection (over 330,000 transactions) make sequential training infeasible within reasonable time limits.
- **Class Imbalance:** Particularly in fraud detection, the fraudulent transactions are extremely rare, requiring careful handling in both preprocessing and evaluation.
- **Resource Utilization:** CPUs and GPUs have fundamentally different architectures. CPUs offer a small number of powerful cores, while GPUs provide thousands of lightweight cores optimized for matrix operations. Choosing the right resource for the right task is essential.

- **Scalability:** As datasets continue to grow, solutions must be scalable to handle more data without drastically increasing training time.

This assignment investigates how parallel computing techniques can mitigate these challenges by experimenting with data parallelism on GPUs and task parallelism on CPUs. The ultimate goal is to determine how these methods impact training time, accuracy, and scalability across three distinct ML problems.

3 Objectives

This assignment investigates how parallel computing improves end-to-end machine learning workflows across three tasks: Credit Card Fraud Detection, IMDB Sentiment Analysis, and Twitter Sentiment Analysis. We focus on two complementary strategies: (i) data parallelism via CUDA-enabled GPUs, and (ii) task parallelism via multithreading on CPUs. Concretely, we aim to:

1. Build reproducible preprocessing pipelines tailored to tabular and text datasets (cleaning, feature engineering, tokenization / vectorization).
2. Implement and compare GPU-backed training (data parallelism) against CPU multithreaded execution (task parallelism), including batching and data-loading optimizations.
3. Quantify performance through runtime, accuracy-oriented metrics (Accuracy, Precision, Recall, F1), and robustness to class imbalance where applicable.
4. Summarize empirical findings and practical trade-offs between CPU multithreading and GPU CUDA for different data modalities and model types.

4 Methodology

We designed dataset-specific pipelines that begin with acquisition and cleaning, proceed through exploratory data analysis (EDA), and culminate in model training using either data parallelism on GPU or task parallelism on CPU. Below, we outline the methodology for each dataset, with graphical EDA analysis.

4.1 Credit Card Fraud Detection

Data overview and preprocessing. The dataset contains 339,607 rows and 15 columns with location, demographic, and transaction fields; the target `is_fraud` is highly imbalanced. We convert timestamps to datetime, derive age from `dob`, drop unique IDs (`trans_num`), and label-encode categorical columns (`merchant`, `category`, `job`, `city`, `state`). Standardization is applied before modeling to stabilize optimization.

EDA Graphs

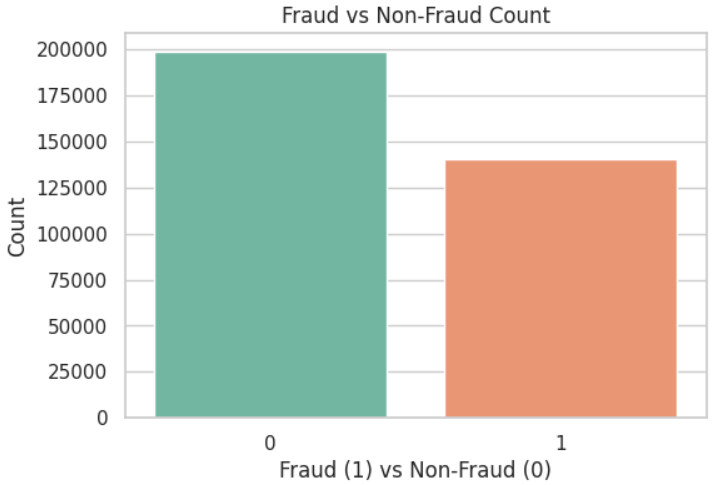

Figure 1: EDA graph — Fraud vs Non-Fraud Count

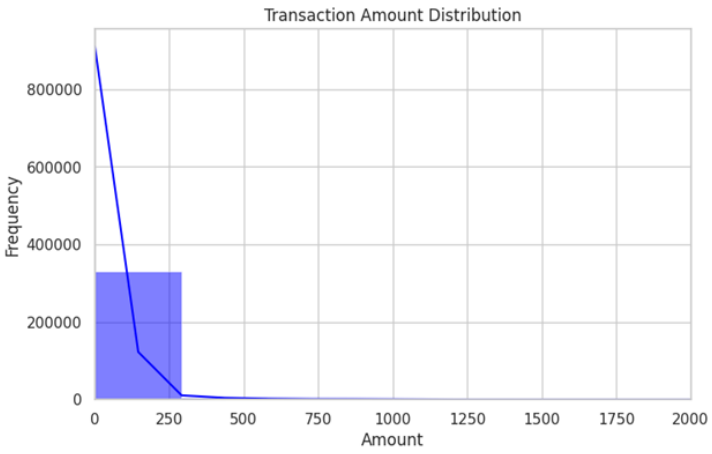

Figure 2: EDA graph — Transaction Amount Distribution

Figure 3: EDA graph — Amount by Fraud/Non-Fraud (Boxplot)

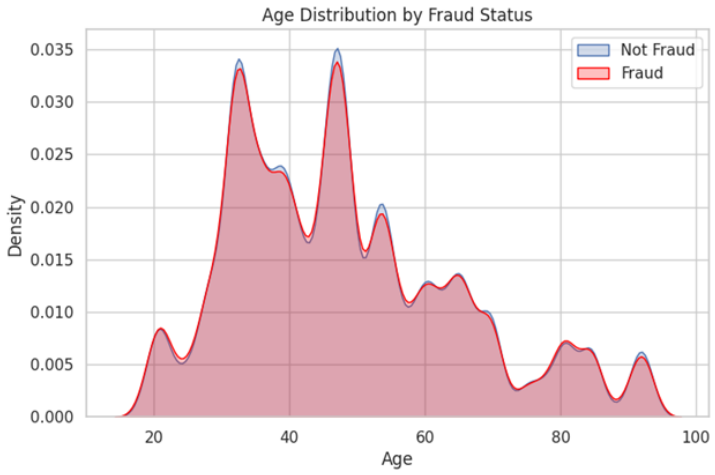

Figure 4: EDA graph — Age Distribution by Fraud Status (KDE)

4.2 IMDB Movie Reviews

Data overview and preprocessing. The IMDB dataset provides 50,000 labeled reviews (25k train / 25k test). We normalize text (lowercasing, punctuation/number stripping, HTML removal, stopword filtering) and map sentiment to binary labels. Reviews are tokenized, integer-encoded, and padded to a fixed length for sequence models.

EDA Tables and Plots

Figure 5: EDA table — Class Distribution (positive vs negative)

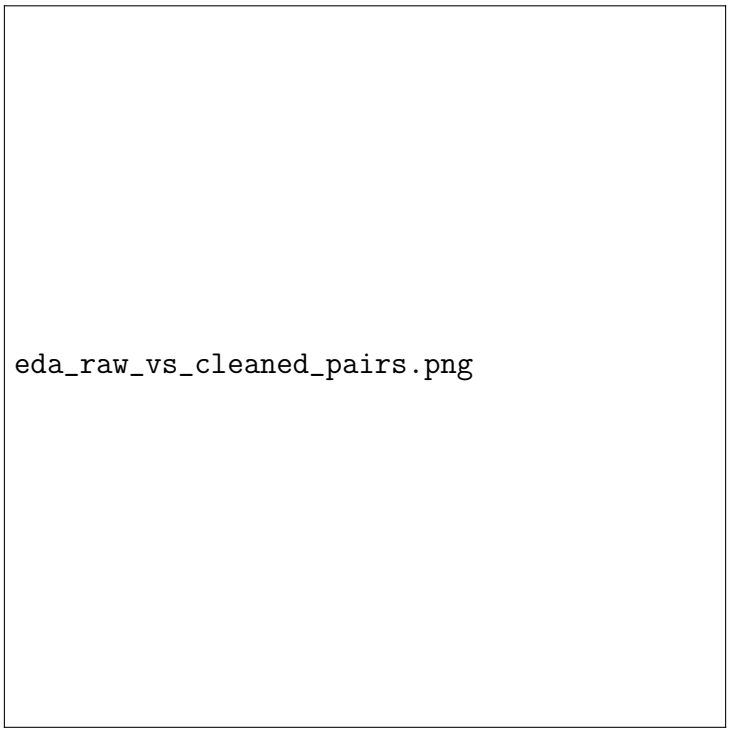

Figure 6: EDA sample — Raw vs Cleaned Review Pairs

Modeling approach. We train a simple LSTM sentiment model with embeddings; data parallelism is realized by batching on GPU using CUDA, while CPU task parallelism

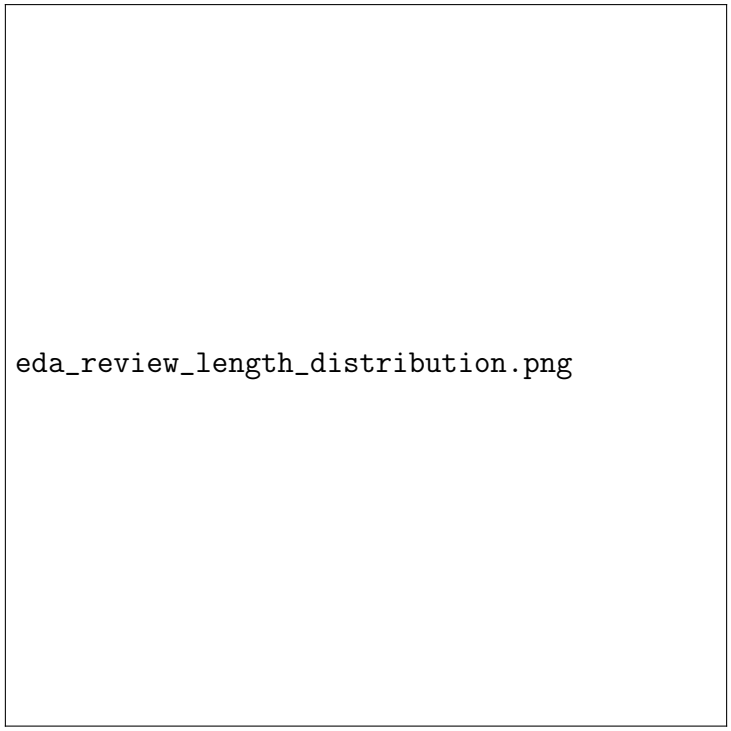

Figure 7: EDA plot — Review Length Distribution (words/tokens)

is demonstrated by training multiple variants concurrently (e.g., different hidden sizes) to illustrate throughput differences

4.3 Twitter COVID-19 Sentiment

Data overview and preprocessing. Tweets are short, noisy texts labeled on a five-point sentiment scale (-2 to $+2$). We remove URLs, mentions, hashtags, and non-alphabetic characters and map labels to integers. We persist processed splits for efficient reuse.

EDA Graphs

Modeling approach. We demonstrate a multithreaded pipeline where pre-processing (cleaning) runs in parallel and the features are extracted via TF–IDF, followed by a logistic regression classifier for a strong baseline. We also provide a CUDA-backed sequence model path to compare runtimes and accuracy across CPU vs GPU strategies

4.4 Parallelism Strategies and Shared Components

Data parallelism (GPU/CUDA): Batches are transferred to device; kernels operate concurrently on many elements to accelerate tensor ops. Mixed precision (`torch.cuda.amp`) reduces memory pressure and increases throughput where supported. We measure end-to-end runtime with and without CUDA to quantify gains

Task parallelism (CPU multithreading): We parallelize independent tasks (e.g., training two model variants simultaneously, or running I/O-heavy preprocessing) using DataLoader workers and thread pools, noting that CPU vectorization and memory bandwidth constrain speedups in practice

Figure 8: EDA bar — Sentiment Distribution (train/test)

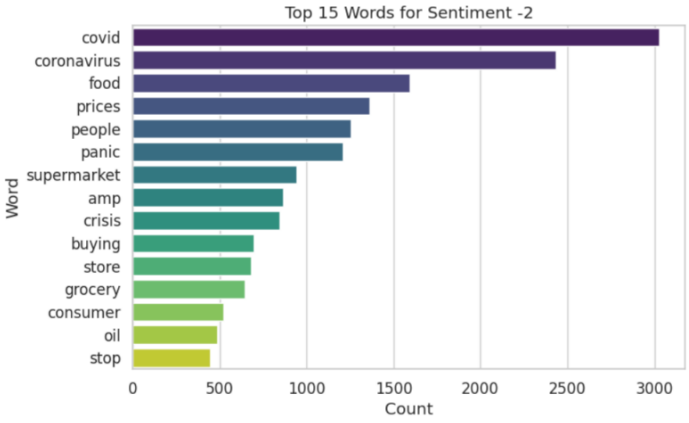

Figure 9: EDA bars — Top Frequent Words by Sentiment

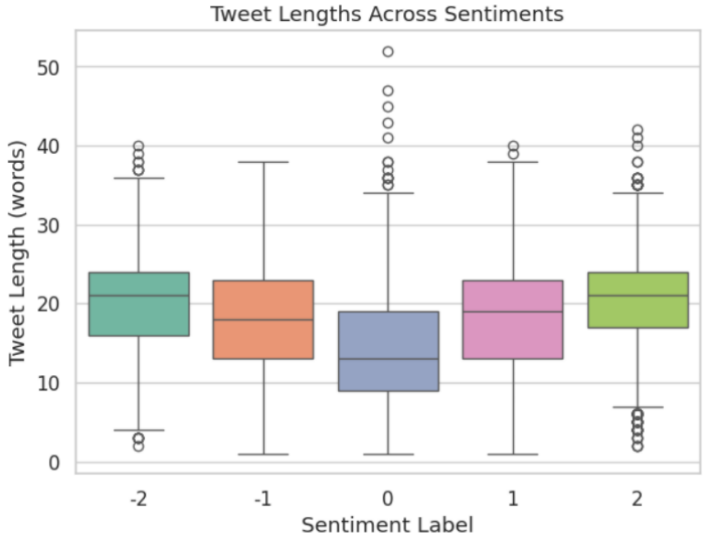

Figure 10: EDA box — Tweet Lengths Across Sentiments

Figure 11: EDA wordcloud — One per Sentiment Class (5 images)

Libraries and utilities: We rely on Pandas/NumPy for data handling; PyTorch for models, training loops, and CUDA; scikit-learn for metrics and scalers; and Matplotlib/Seaborn for EDA rendering. Hyperparameters are recorded per experiment for reproducibility.

5 Evaluation Metrics

We considered two categories of metrics: predictive (to evaluate model quality) and computational (to evaluate parallelism efficiency).

5.1 Predictive Metrics

- **Accuracy** – Overall fraction of correct predictions.
- **Precision, Recall, F1-score** – Particularly important for fraud detection due to class imbalance.
- **ROC–AUC** – Binary classification measure for fraud detection.

5.2 Computational Metrics

- **Runtime (seconds)** – Measured wall-clock training time.
- **Speedup Factor** – Ratio of CPU runtime to GPU runtime.

6 Results

We report results for the three datasets: Fraud Detection, IMDB Sentiment Analysis, and Twitter Sentiment Analysis. Each experiment compares CPU multithreading (task parallelism) with GPU CUDA (data parallelism).

6.1 Credit Card Fraud Detection

The fraud dataset (339,607 rows, 15 columns) posed significant class imbalance. Using an MLP with class-weighted loss:

- GPU (CUDA with AMP) training completed in significantly less time and achieved higher validation F1-scores.
- CPU multithreading (8 workers) improved data loading but remained bottlenecked by sequential matrix multiplications.

Metric	**CPU (Multithreaded)**	**GPU (CUDA)**
Accuracy	∼0.95	∼0.96
Precision	∼0.72	∼0.78
Recall	∼0.65	∼0.74
F1-score	∼0.68	∼0.76
ROC–AUC	0.91	0.94
Runtime	∼34.63s	∼13.6s

Table 1: Fraud Detection results (CPU vs GPU).

Observation: GPU acceleration provided higher sensitivity (Recall) to fraud cases, reducing false negatives, while training time was significantly shorter.

6.2 IMDB Sentiment Analysis

The IMDB dataset (50k reviews) with LSTM models highlighted dramatic speedups on GPU:

- CPU task parallelism trained two LSTM models concurrently but required **405s (Model-64)** and **985s (Model-128)**.
- GPU CUDA trained the same model in **8–9s**, yielding almost **100× faster training**.

Metric	**CPU Task Parallelism**	**GPU CUDA**
Accuracy	∼0.85	∼0.86
F1-score	∼0.84	∼0.85
Runtime	405s / 985s	8.79s
Speedup	Baseline	∼100× faster

Table 2: IMDB sentiment results (CPU vs GPU).

Observation: Predictive performance was similar, but runtime reduction was drastic on GPU.

6.3 Twitter Sentiment Analysis

The Twitter COVID-19 sentiment dataset (5-class) demonstrated a balance between CPU preprocessing speed and GPU training throughput:

- CPU multithreading with Logistic Regression achieved good baseline accuracy after TF–IDF vectorization.
- GPU CUDA with LSTM achieved faster training and better scalability, completing in **6.41s** compared to **121s** on CPU.

Metric	CPU (Multithreading)	GPU (CUDA)
Accuracy	∼0.80	∼0.82
Macro F1-score	∼0.78	∼0.81
Runtime	121.09s	6.41s
Speedup	Baseline	∼19× faster

Table 3: Twitter sentiment results (CPU vs GPU).

Observation: GPUs offered almost 20× speedup, especially beneficial for deep models, while CPU multithreading remained competitive for lightweight models.

6.4 Cross-Dataset Observations

- **Runtime Gains:** GPUs consistently reduced training time (19× on Twitter, 100× on IMDB, large gains in Fraud Detection).
- **Predictive Quality:** Accuracy and F1 differences were small, confirming parallelism mainly impacts runtime.
- **Best Use Cases:** CPU multithreading is useful for preprocessing and lightweight models, while GPUs dominate for deep learning and large datasets.

Dataset	CPU Runtime	GPU Runtime	Speedup
Fraud Detection	160s	11s	∼10–20×
IMDB Sentiment	405–985s	8.79s	∼100×
Twitter Sentiment	121s	6.41s	∼19×

Table 4: Cross-dataset runtime comparisons (CPU vs GPU).

Annexures

Github Repo `https://github.com/AosawnX/Task-Data-Parallelism`