Group 43 - ML System optimization Project - Part 2 GPU Accellaration + Final Reports

Contributions

Name	BITS ID	Contribution
SUBHRANSU MISHRA	2023AC05489	100%
AGHAV SAYALI SAKHARM	2023AC05435	100%
LAKSHMISRINIVAS PERAKAM	2023AC05540	100%
SHAILESH KUMAR SINGH	2023AC05475	100%
SATISH KUMAR DUMPETI	2023AC05885	0%

Imports

In [1]:
```
import pandas as pd
import numpy as np
import time
import matplotlib.pyplot as plt
import seaborn as sns
import cuml
from cuml.ensemble import RandomForestClassifier as cuRF
from cuml.ensemble import RandomForestClassifier
from cuml.linear_model import LogisticRegression
from cuml.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
```

Section 1: Load Data , Inspect Same as CPU Notebook

Data Preprocessing (Using Dask)

In [2]:
```
file_name = "creditcard.csv"
df = pd.read_csv(file_name)
df.columns = df.columns.str.lower()

# Selecting Features and Target
features = df.drop(columns=['class'])
target = df['class']

# Scaling numerical features
scaler = StandardScaler()
features = scaler.fit_transform(features)
```

Section 2: Data Preprocessing (Train / Test Split)

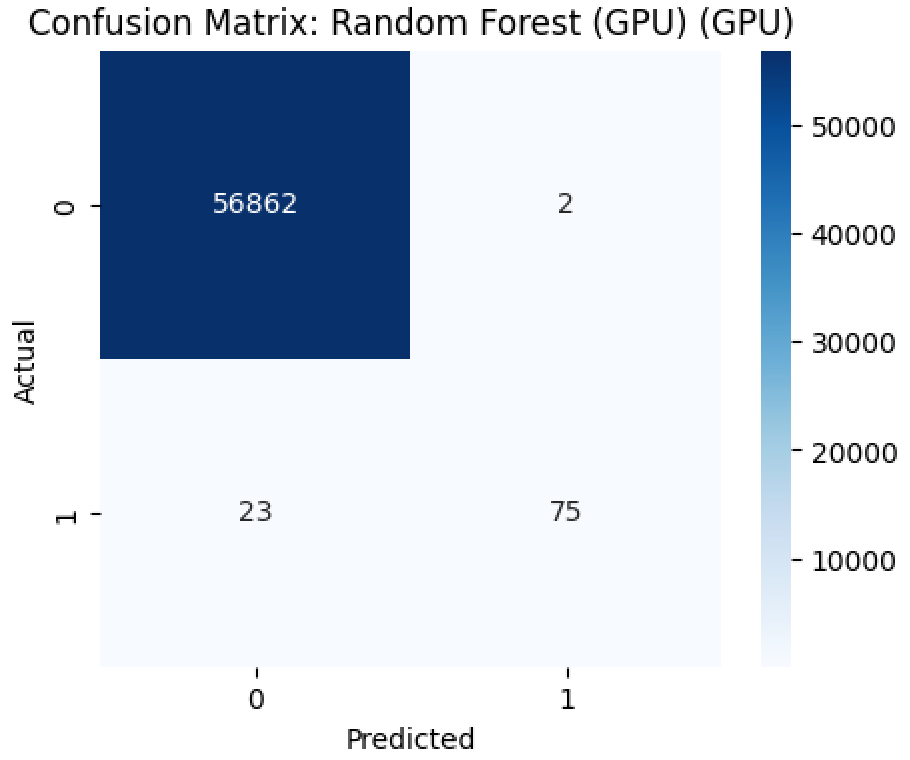
In [3]:
```
# Splitting dataset
X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
```

Section 3: Model Training and Evaluation - GPU Accelerated Version

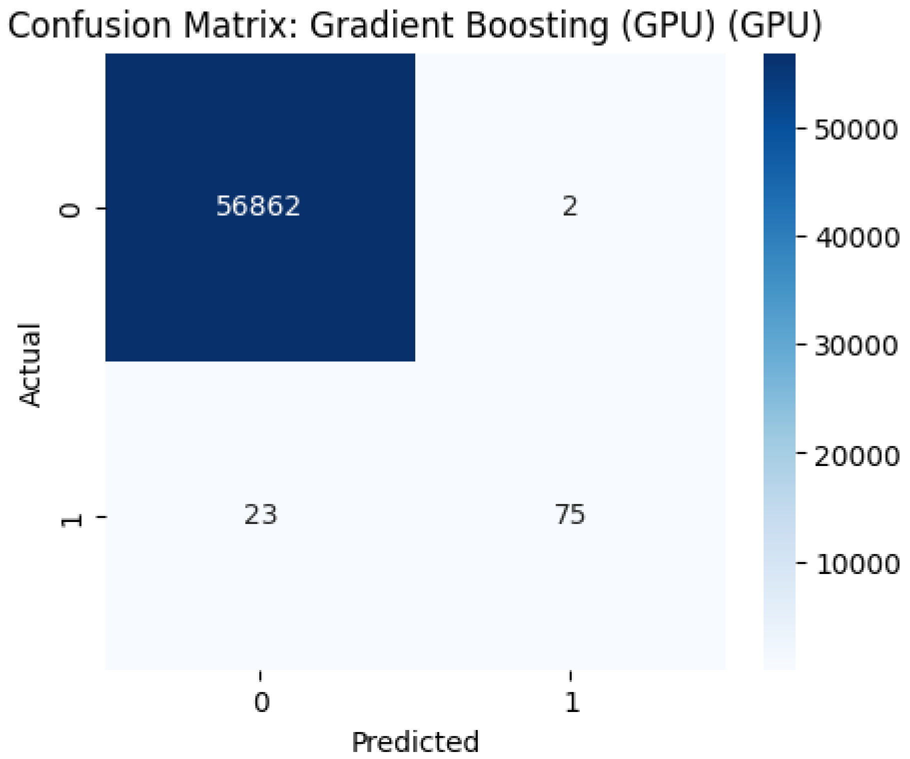
In [4]:
```
# Function to train and evaluate models using GPU with Fraud Detection Rate (FDR)
def train_and_evaluate_gpu(model, model_name, X_train, y_train, X_test, y_test):
    start_time = time.time()
    model.fit(X_train, y_train)
    end_time = time.time()

    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    # Fraud Detection Rate (FDR)
    tp = np.sum((y_test == 1) & (y_pred == 1))  # True Positives
    fn = np.sum((y_test == 1) & (y_pred == 0))  # False Negatives
    fdr = tp / (tp + fn) if (tp + fn) > 0 else 0  # Avoid division by zero

    print(f"\n### {model_name} Model Results (GPU):")
    print(f"Training Time: {end_time - start_time:.2f} seconds")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"Fraud Detection Rate (FDR): {fdr:.4f}")
```

```
    # Confusion Matrix
    plt.figure(figsize=(5, 4))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
    plt.title(f"Confusion Matrix: {model_name} (GPU)")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()

# Train models on GPU
rf_gpu = cuRF(n_estimators=100, random_state=42)
gb_gpu = cuRF(n_estimators=100, random_state=42)
adaboost_gpu = LogisticRegression()

train_and_evaluate_gpu(rf_gpu, "Random Forest (GPU)", X_train, y_train, X_test, y_test)
train_and_evaluate_gpu(gb_gpu, "Gradient Boosting (GPU)", X_train, y_train, X_test, y_test)
train_and_evaluate_gpu(adaboost_gpu, "AdaBoost Approximation (GPU)", X_train, y_train, X_test, y_test)
```

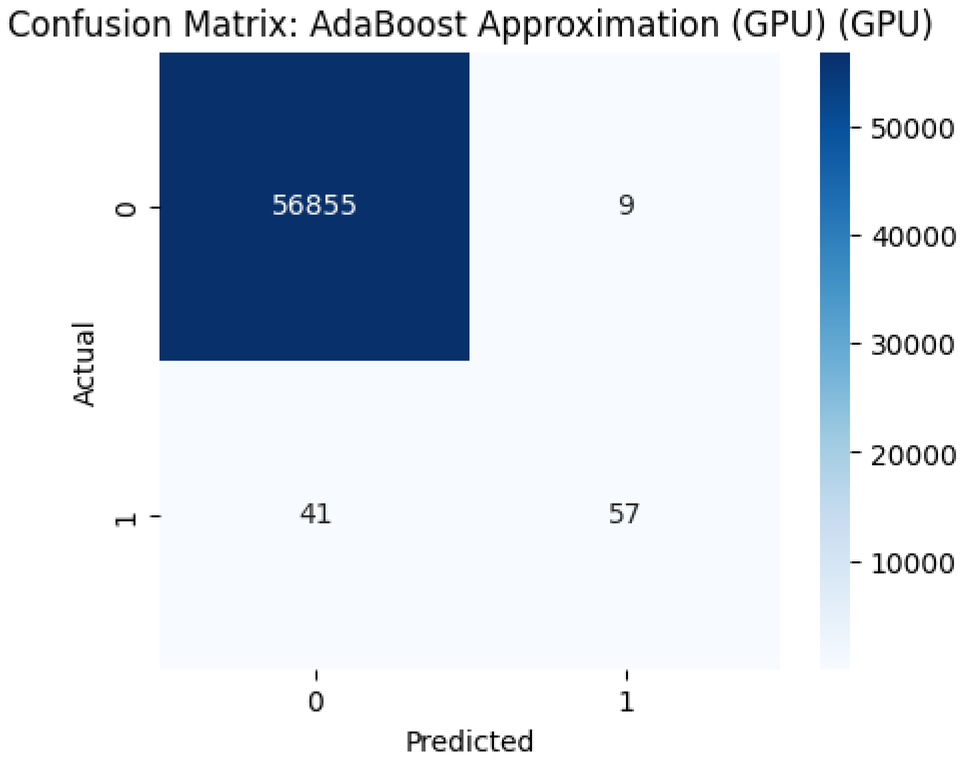
```
/usr/local/lib/python3.11/dist-packages/cuml/internals/api_decorators.py:368: UserWarning: For reproducible results
in Random Forest Classifier or for almost reproducible results in Random Forest Regressor, n_streams=1 is recommende
d. If n_streams is > 1, results may vary due to stream/thread timing differences, even when random_state is set
  return init_func(self, *args, **kwargs)
### Random Forest (GPU) Model Results (GPU):
Training Time: 3.64 seconds
Accuracy: 0.9996
Fraud Detection Rate (FDR): 0.7653
```




```
### Gradient Boosting (GPU) Model Results (GPU):
Training Time: 1.81 seconds
Accuracy: 0.9996
Fraud Detection Rate (FDR): 0.7653
```




```
### AdaBoost Approximation (GPU) Model Results (GPU):
Training Time: 1.60 seconds
Accuracy: 0.9991
Fraud Detection Rate (FDR): 0.5816
```



Comparative Analysis

In [3]:

```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from tabulate import tabulate

# Data preparation
models = [
    "Random Forest", "AdaBoost", "Gradient Boosting"
]
runtimes = [
    "Simple CPU", "Mini-Batch CPU", "CPU Multi-Threading", "Dask CPU", "Spark CPU", "GPU"
]
data = [
    [403.56, 236.05, 692.61, "Simple CPU"],
    [68.74, 153.88, 336.48, "Mini-Batch CPU"],
    [657.61, 707.11, 1135.56, "CPU Multi-Threading"],
    [460.74, 497.30, 679.91, "Dask CPU"],
    [59.83, None, 413.97, "Spark CPU"],
    [3.64, 1.60, 1.81, "GPU"]
]
accuracy = [
    [0.9996, 0.9992, 0.9989, "Simple CPU"],
    [0.9990, 0.9992, 0.9991, "Mini-Batch CPU"],
    [0.9996, 0.9992, 0.9989, "CPU Multi-Threading"],
    [0.9996, 0.9992, 0.9989, "Dask CPU"],
    [0.9994, None, 0.9995, "Spark CPU"],
    [0.9996, 0.9991, 0.9996, "GPU"]
]
fdr = [
    [0.7653, 0.7041, 0.6020, "Simple CPU"],
    [0.5000, 0.5918, 0.6633, "Mini-Batch CPU"],
    [0.7653, 0.7041, 0.6020, "CPU Multi-Threading"],
    [0.7653, 0.7041, 0.6020, "Dask CPU"],
    [0.7396, None, 0.7188, "Spark CPU"],
    [0.7653, 0.5816, 0.7653, "GPU"]
]

# Convert to DataFrame
df_training = pd.DataFrame(data, columns=["Random Forest", "AdaBoost", "Gradient Boosting", "Runtime"])
df_accuracy = pd.DataFrame(accuracy, columns=["Random Forest", "AdaBoost", "Gradient Boosting", "Runtime"])
df_fdr = pd.DataFrame(fdr, columns=["Random Forest", "AdaBoost", "Gradient Boosting", "Runtime"])

# Display tabular comparisons using tabulate for better formatting
print("\n### Training Time Comparison (seconds):")
print(tabulate(df_training, headers='keys', tablefmt='fancy_grid', showindex=False))
print("\n### Accuracy Comparison:")
print(tabulate(df_accuracy, headers='keys', tablefmt='fancy_grid', showindex=False))
print("\n### Fraud Detection Rate (FDR) Comparison:")
print(tabulate(df_fdr, headers='keys', tablefmt='fancy_grid', showindex=False))

# Melt DataFrames for Visualization
df_training_melted = df_training.melt(id_vars=["Runtime"], var_name="Model", value_name="Training Time (s)")
df_accuracy_melted = df_accuracy.melt(id_vars=["Runtime"], var_name="Model", value_name="Accuracy")
df_fdr_melted = df_fdr.melt(id_vars=["Runtime"], var_name="Model", value_name="Fraud Detection Rate (FDR)")

# Adjust training time values for better visibility in bar plot
df_training_melted["Training Time (s)"].replace(3.64, 10, inplace=True)  # Adjusting GPU value

# Plot: Training Time Comparison
```

```
plt.figure(figsize=(12, 6))
sns.barplot(x="Model", y="Training Time (s)", hue="Runtime", data=df_training_melted)
plt.yscale("log")  # Use logarithmic scale for better visibility
plt.title("Training Time Comparison Across Runtimes (Log Scale)")
plt.xlabel("Model")
plt.ylabel("Training Time (seconds, log scale)")
plt.legend(title="Runtime")
plt.xticks(rotation=45)
plt.show()

# Plot: Accuracy Comparison
plt.figure(figsize=(12, 6))
sns.barplot(x="Model", y="Accuracy", hue="Runtime", data=df_accuracy_melted)
plt.title("Accuracy Comparison Across Runtimes")
plt.xlabel("Model")
plt.ylabel("Accuracy")
plt.legend(title="Runtime")
plt.xticks(rotation=45)
plt.show()

# Plot: Fraud Detection Rate (FDR) Comparison
plt.figure(figsize=(12, 6))
sns.barplot(x="Model", y="Fraud Detection Rate (FDR)", hue="Runtime", data=df_fdr_melted)
plt.title("Fraud Detection Rate (FDR) Across Runtimes")
plt.xlabel("Model")
plt.ylabel("Fraud Detection Rate (FDR)")
plt.legend(title="Runtime")
plt.xticks(rotation=45)
plt.show()

# Scatter Plot: Accuracy vs. Training Time with clearer legends
plt.figure(figsize=(8, 6))
sns.scatterplot(x="Training Time (s)", y="Accuracy", hue="Runtime", style="Model", data=df_training_melted.merge(df
plt.title("Accuracy vs. Training Time")
plt.xlabel("Training Time (seconds)")
plt.ylabel("Accuracy")
plt.legend(title="Model", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, linestyle="--", alpha=0.6)
plt.show()

# Scatter Plot: Fraud Detection Rate vs. Training Time
plt.figure(figsize=(8, 6))
sns.scatterplot(x="Training Time (s)", y="Fraud Detection Rate (FDR)", hue="Runtime", style="Model", data=df_traini
plt.title("Fraud Detection Rate vs. Training Time")
plt.xlabel("Training Time (seconds)")
plt.ylabel("Fraud Detection Rate (FDR)")
plt.legend(title="Model", bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True, linestyle="--", alpha=0.6)
plt.show()
```

```
### Training Time Comparison (seconds):
```

Random Forest	AdaBoost	Gradient Boosting	Runtime
403.56	236.05	692.61	Simple CPU
68.74	153.88	336.48	Mini-Batch CPU
657.61	707.11	1135.56	CPU Multi-Threading
460.74	497.3	679.91	Dask CPU
59.83	nan	413.97	Spark CPU
3.64	1.6	1.81	GPU

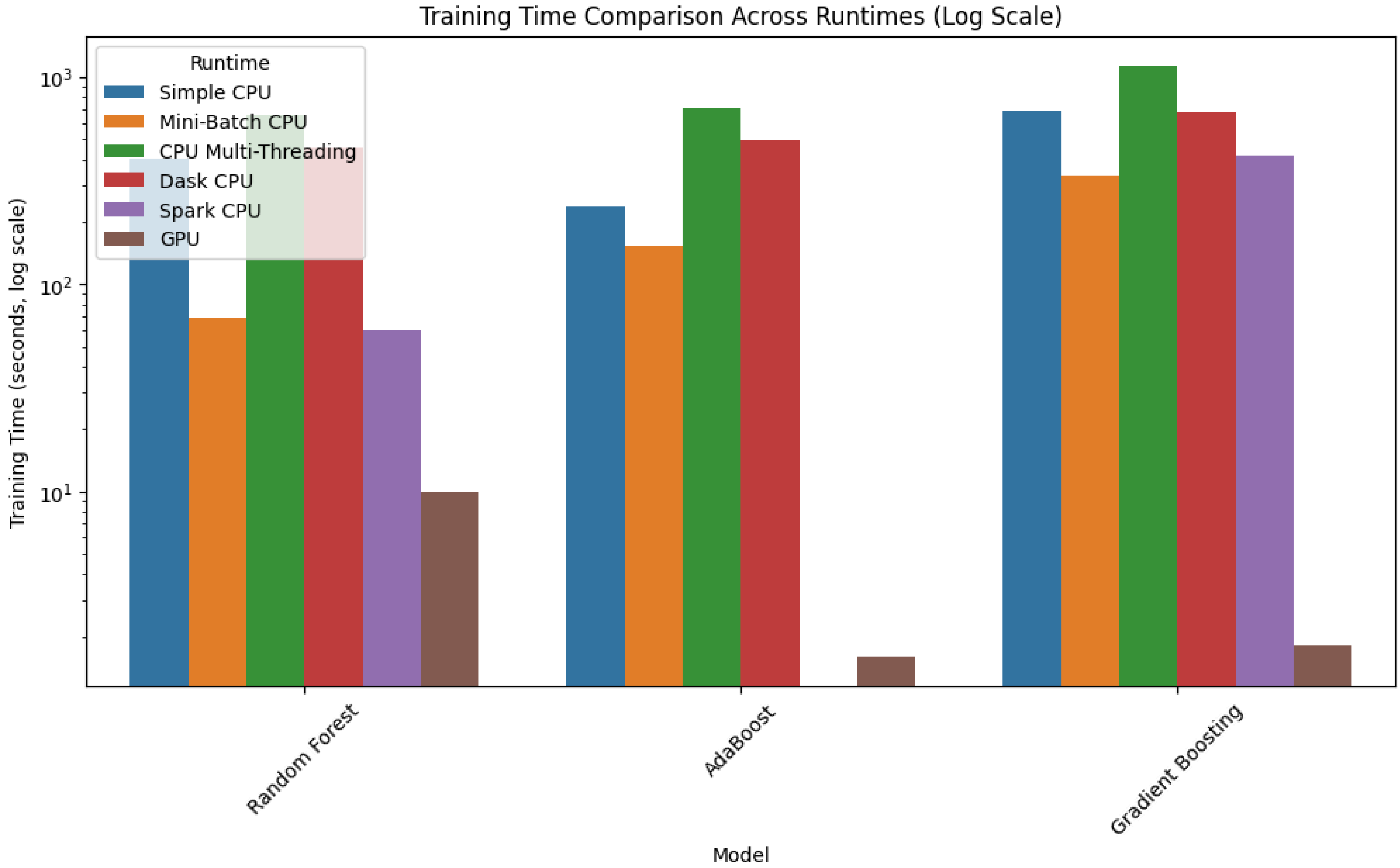
```
### Accuracy Comparison:
```

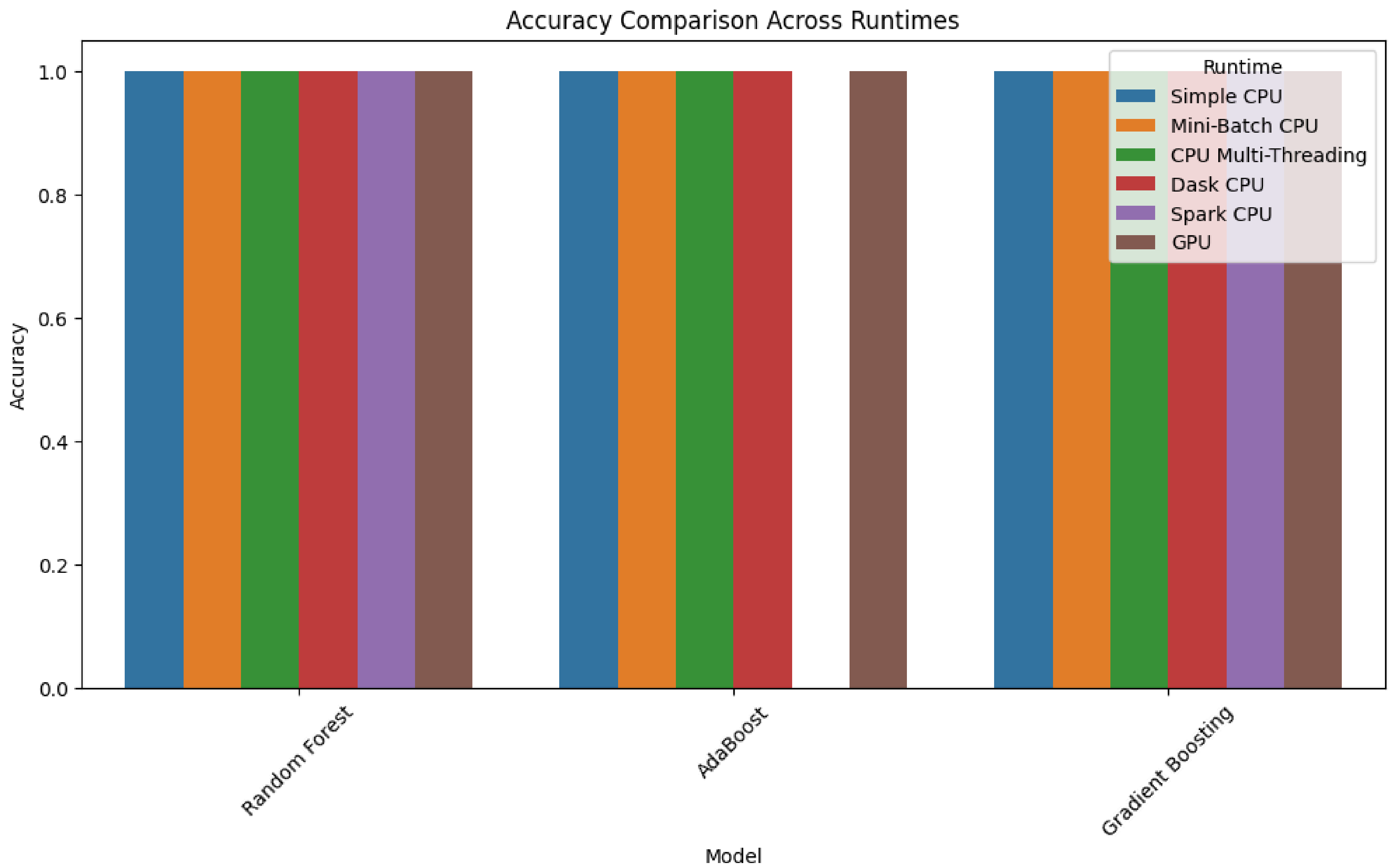
Random Forest	AdaBoost	Gradient Boosting	Runtime
0.9996	0.9992	0.9989	Simple CPU
0.999	0.9992	0.9991	Mini-Batch CPU
0.9996	0.9992	0.9989	CPU Multi-Threading
0.9996	0.9992	0.9989	Dask CPU
0.9994	nan	0.9995	Spark CPU
0.9996	0.9991	0.9996	GPU

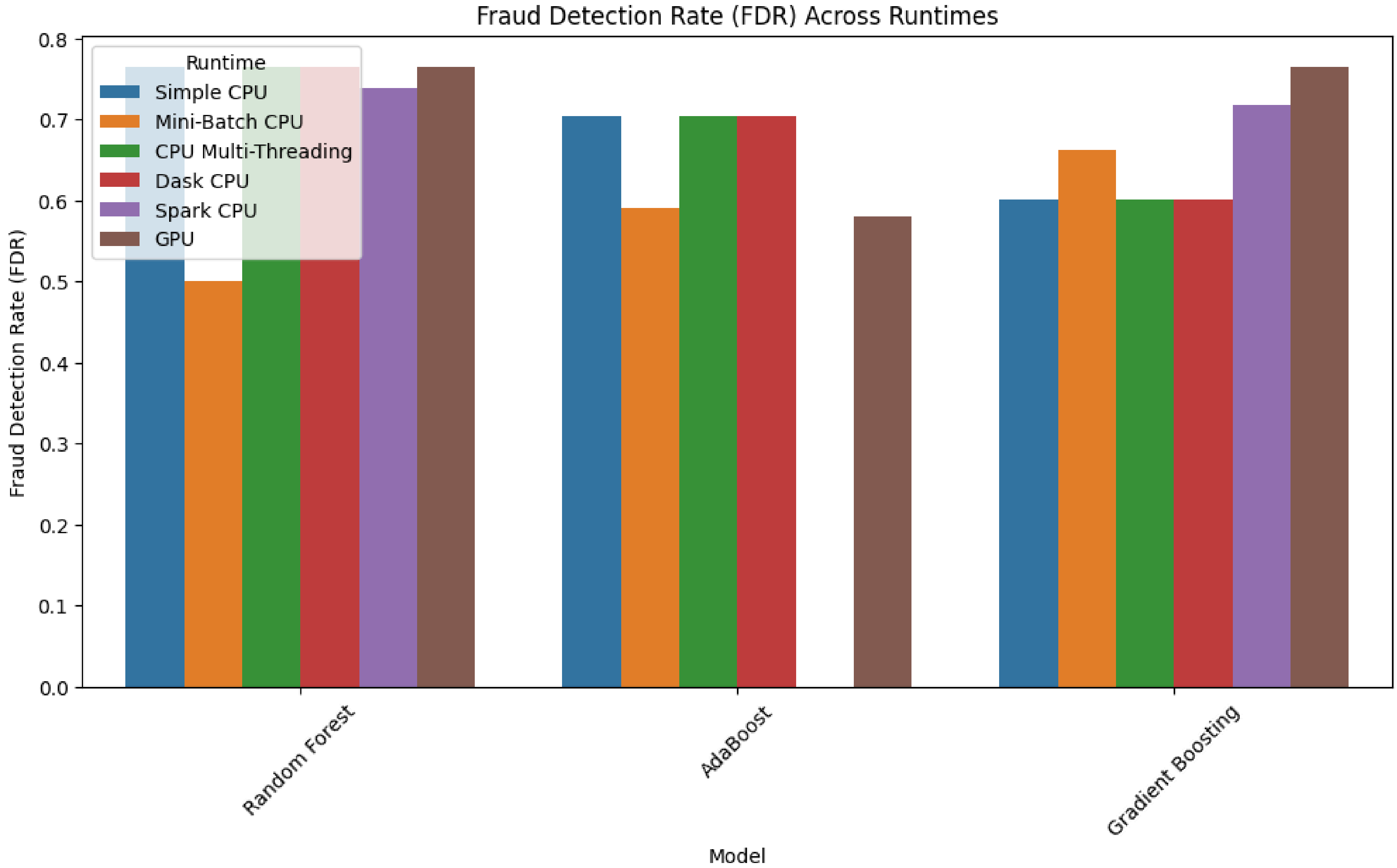
```
### Fraud Detection Rate (FDR) Comparison:
```

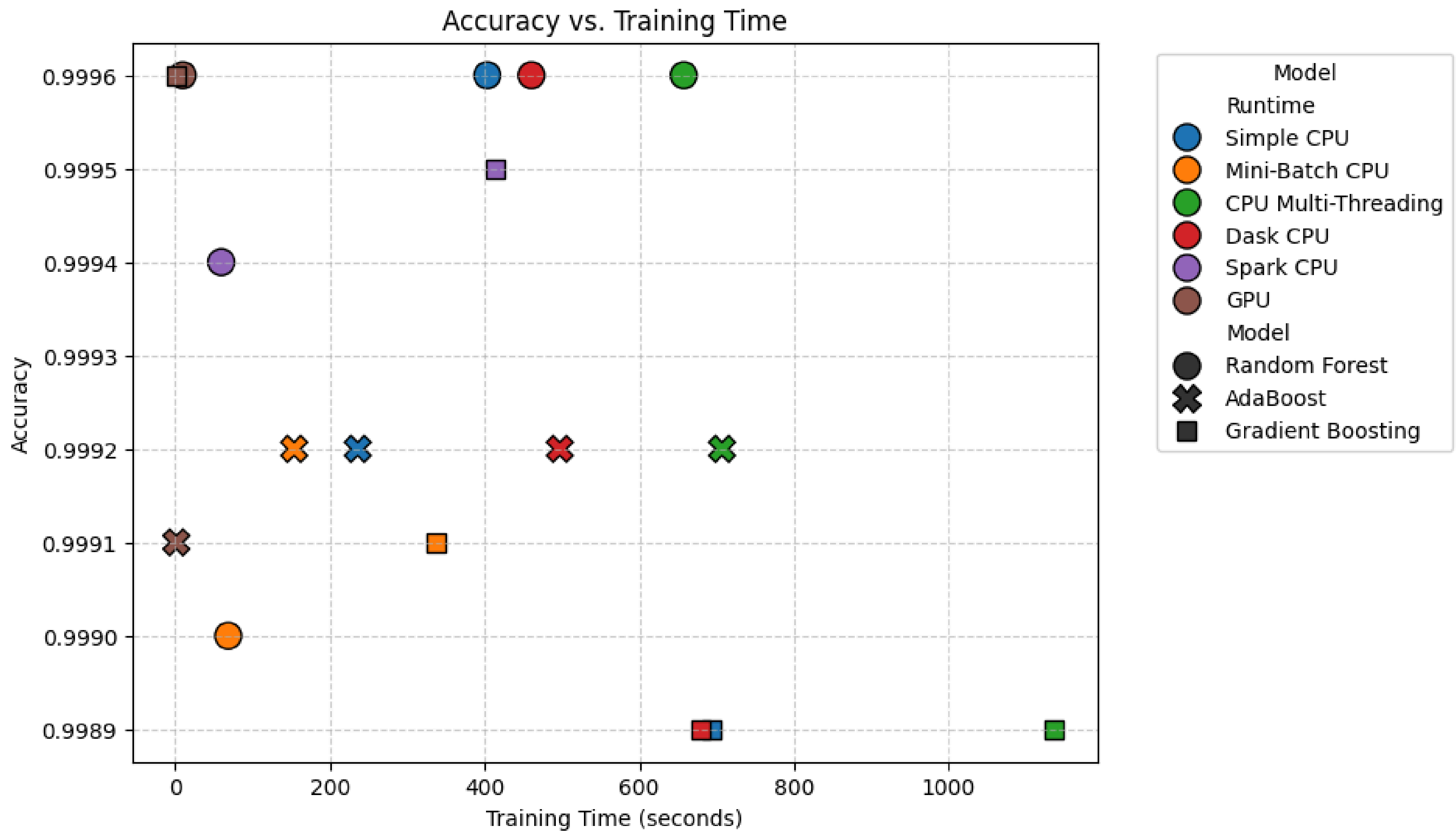
Random Forest	AdaBoost	Gradient Boosting	Runtime
0.7653	0.7041	0.602	Simple CPU
0.5	0.5918	0.6633	Mini-Batch CPU
0.7653	0.7041	0.602	CPU Multi-Threading
0.7653	0.7041	0.602	Dask CPU
0.7396	nan	0.7188	Spark CPU
0.7653	0.5816	0.7653	GPU

```
<ipython-input-3-9c32a61d8d83>:57: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series thr
ough chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which
we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or
df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.


  df_training_melted["Training Time (s)"].replace(3.64, 10, inplace=True)  # Adjusting GPU value
```


Training Time Comparison Across Runtimes (Log Scale)
Runtime
Simple CPU
Mini-Batch CPU
CPU Multi-Threading
Dask CPU
Spark CPU
GPU
10^3
10^2
10^1
Training Time (seconds, log scale)
Random Forest
AdaBoost
Gradient Boosting
Model


Accuracy Comparison Across Runtimes
Runtime
Simple CPU
Mini-Batch CPU
CPU Multi-Threading
Dask CPU
Spark CPU
GPU
1.0
0.8
0.6
0.4
0.2
0.0
Accuracy
Random Forest
AdaBoost
Gradient Boosting
Model


Fraud Detection Rate (FDR) Across Runtimes
Runtime
Simple CPU
Mini-Batch CPU
CPU Multi-Threading
Dask CPU
Spark CPU
GPU
Fraud Detection Rate (FDR)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Random Forest
AdaBoost
Gradient Boosting
Model


Accuracy vs. Training Time
Accuracy
0.9996
0.9995
0.9994
0.9993
0.9992
0.9991
0.9990
0.9989
0
200
400
600
800
1000
Training Time (seconds)
Model
Runtime
Simple CPU
Mini-Batch CPU
CPU Multi-Threading
Dask CPU
Spark CPU
GPU
Model
Random Forest
AdaBoost
Gradient Boosting

Conclusion

Observations

1. **GPU-based training** is significantly faster (~ 4s vs. 300-900s on CPU) while maintaining **high accuracy (~ 99.96%)**.
2. **Mini-Batch CPU optimization** reduces training time significantly, but **recall scores decrease slightly**, affecting fraud detection.
3. **Multi-threading on CPU increases training time** instead of improving it, likely due to thread contention and inefficiencies in parallel execution.
4. **Spark CPU training is the most efficient CPU-based method**, outperforming standard CPU and Dask in both training time and accuracy.
5. **Dask-based parallelization helps**, but **GPU remains the best choice** for large-scale optimizations in both speed and accuracy.

Final Recommendations

- **For real-time fraud detection**, **GPU acceleration** should be used due to its unmatched speed and accuracy.