QUANTUM COMPUTING INC

Profiling of CVQBoost Algorithm: Fraud Detection

February 2025

Quantum Computing Inc
quantumcomputinginc.com
(703) 436-2161

Abstract

This study evaluates the performance of CVQBoost, a novel extension of the QBoost algorithm leveraging quadratic optimization solvers like QCi's Dirac-3, for classification tasks. The performance of CVQBoost is benchmarked against the state-of-the-art XGBoost algorithm across different datasets.

First, we evaluate CVQBoost in a fraud detection scenario using the imbalanced Kaggle Credit Card Fraud Detection dataset. Our results demonstrate that CVQBoost achieves superior runtime performance, particularly as the size of the training data and the number of features increase, where its scaling advantage becomes increasingly evident. In terms of accuracy, assessed using the Area Under the Curve (AUC), XGBoost initially exhibits higher performance when balancing is minimal, and the minority-to-majority class ratio remains extremely low. However, as the training data becomes more balanced, CVQBoost consistently catches up and achieves comparable AUC scores. Notably, with the ADASYN balancing strategy, CVQBoost eventually surpasses XGBoost in accuracy for higher class ratios.

Additionally, we extend our study to a large-scale synthetic dataset generated using the Scikit-learn package. This dataset, ranging from 1M to 70M samples, is used to compare the scalability of CVQBoost against XGBoost running on 48 CPUs and four NVIDIA L4 GPUs under the NVIDIA RAPIDS AI environment. Results indicate that CVQBoost scales much better than both multi-core CPU-based XGBoost and GPU-accelerated XGBoost, demonstrating its suitability for large-scale data.

In summary, benchmarking results show that CVQBoost on Dirac-3 provides superior scalability while maintaining competitive accuracy, making it a promising alternative for both fraud detection and large-scale classification problems.

1 Introduction

Boosting [1] is a powerful machine learning technique that combines multiple weak classifiers into a strong classifier. QBoost [2] formulates boosting as a quadratic unconstrained binary optimization (QUBO) problem, making it suitable for quantum solvers. CVQBoost extends QBoost by leveraging continuous-variable quantum optimization, particularly on QCi's Dirac-3 platform.

We evaluate CVQBoost in two different contexts. First, we apply it to the Kaggle Credit Card Fraud Detection dataset, comparing its performance with XGBoost under different levels of class imbalance and data balancing strategies. Second, we benchmark CVQBoost against XGBoost on large-scale synthetic datasets ranging from 1M to 70M samples, assessing its scalability on modern multi-core CPUs and GPUs.

In the fraud detection scenario, we examine CVQBoost's runtime performance, scalability, and classification accuracy under varying dataset sizes and feature counts. We also assess its accuracy using the Area Under the Curve (AUC) metric across multiple class balancing strategies, including ADASYN, SMOTE, and downsampling.

For the synthetic dataset, we investigate the scalability of CVQBoost in comparison to XGBoost running on 48 CPU cores and four NVIDIA L4 GPUs under the NVIDIA RAPIDS AI framework. The results demonstrate that CVQBoost scales significantly better than both CPU-based and GPU-accelerated XGBoost, particularly as dataset size increases.

Overall, our findings highlight CVQBoost's advantages in terms of computational efficiency, scalability, and competitive classification accuracy, making it a strong candidate for both imbalanced and large-scale classification problems.

2 Formulation

Our methods are based on a fairly conventional formulation of boosting [1], most similar to the quadratic formulation used in QBoost[2]. Let us assume that we have a collection of N "weak" classifiers h_i where $i = 1, 2, ..., N$. Depending on the method used to build weak classifiers, the values of h_i can be discrete or continuous. The goal is to construct a "strong" classifier as a linear superposition of these weak classifiers, that is,

$$y = \sum_{i=1}^{N} w_i h_i(\mathbf{x}) \tag{1}$$

where $\mathbf{x}$ is a vector of input features and $y \in \{-1, 1\}$. The goal is to find w_i, weights associated with the weak classifiers.

We use a training set $\{(\mathbf{x_s}, y_s) | s = 1, 2, ..., S\}$ of size S. We can determine optimal weights $w_i \geq 0$ by minimizing,

$$\min_{\mathbf{w}} \sum_{s=1}^{S} \left| \sum_{i=1}^{N} w_i h_i(\mathbf{x_s}) - y_s \right|^2 + \lambda \sum_{i=1}^{N} (w_i)^2 \tag{2}$$

where the regularization term $\lambda \sum_{i=1}^{N} (w_i)^2$ penalizes non-zero weights; λ is the regularization coefficient. Note that this is similar to the formulation in the QBoost [2] algorithm. The key difference is that we do not need to encode the weights in to binary variables since the Dirac machine we use is based on a native continuum encoding.

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \sum_{j=1}^{N} J_{ij} w_i w_j + \sum_{i=1}^{N} C_i w_i \tag{3}$$

where

$$J_{ij} = \sum_{s=1}^{S} h_i(\mathbf{x_s}) h_j(\mathbf{x_s}) \tag{4}$$

and

$$C_i = -2 \sum_{s=1}^{S} y_s h_i(\mathbf{x_s}) \tag{5}$$

subject to (applied as a built in constraint on Dirac-3),

$$\sum_{i=1}^{N} w_i = 1 \tag{6}$$

Note that the above algorithm assumes that the total number of weak classifiers, that is N, is less than the number of available qudits on Dirac-3.

2.1 Choices of Weak Classifiers

There are many ways to design a subset of weak classifiers. We have tested CVQBoost using logistic regression, decision tree, naive Bayesian, and Gaussian process classifiers. Each weak classifier is constructed using one or two of the features chosen from all features. This yields a set of weak classifiers that can be used to construct a strong classifier.

3 Fraud Detection Use Case

3.1 Dataset

The dataset used in this study is the Kaggle Credit Card Fraud Detection dataset [3], a widely recognized resource for machine learning research, particularly in anomaly detection.

This dataset comprises transactions made by European credit cardholders over a two-day period in September 2013. It contains 284,807 transactions, of which 492 (approximately 0.172%) are labeled as fraudulent. The significant class imbalance makes it ideal for exploring techniques tailored to imbalanced classification problems.

A total of 38 features are included in the dataset.

3.2 Time Profiling

3.2.1 Impact of Training Data Count on Runtimes

Table 1 presents the runtimes of CVQBoost and XGBoost across varying training dataset sizes, ranging from 1,000 to 150,000 samples. All runtimes are reported in seconds, with each experiment repeated multiple times to ensure reliability. The table provides both the mean and standard deviation of the runtimes, as well as a breakdown of CVQBoost's runtime components. Notably, the Dirac-3 component accounts for a substantial portion of the total runtime for CVQBoost.

A comparison between CVQBoost and XGBoost reveals that XGBoost achieves shorter runtimes on smaller datasets. However, as the training dataset size increases, XGBoost's runtime grows significantly faster than CVQBoost's. This trend is further illustrated in Figure 1, which plots training runtimes against dataset size.

These results suggest that while XGBoost excels in speed for smaller datasets, CVQBoost offers a clear performance advantage as dataset sizes increase. Specifically, CVQBoost begins to outperform XGBoost when the training sample size approaches 50,000 samples, highlighting its scalability for larger datasets.

train data count	cvqboost train time	xgboost train time	dirac3 time
1000	7.3 +/- 1.3	0.9 +/- 0.1	1.3 +/- 0.5
2000	7.9 +/- 1.3	1.8 +/- 0.1	1.7 +/- 0.5
5000	7.4 +/- 1.2	3.7 +/- 0.1	1.4 +/- 0.5
10000	7.3 +/- 1.1	7.2 +/- 0.2	1.1 +/- 0.3
20000	9.0 +/- 1.3	14.6 +/- 0.3	1.7 +/- 0.9
50000	10.5 +/- 3.2	37.4 +/- 1.2	2.3 +/- 3.1
100000	12.3 +/- 1.1	78.0 +/- 2.9	1.2 +/- 0.4
150000	15.1 +/- 1.4	117.4 +/- 4.0	1.3 +/- 0.5

Table 1: Breakdown of CVQBoost and XGBoost runtimes for different training data counts; number of features: 38.

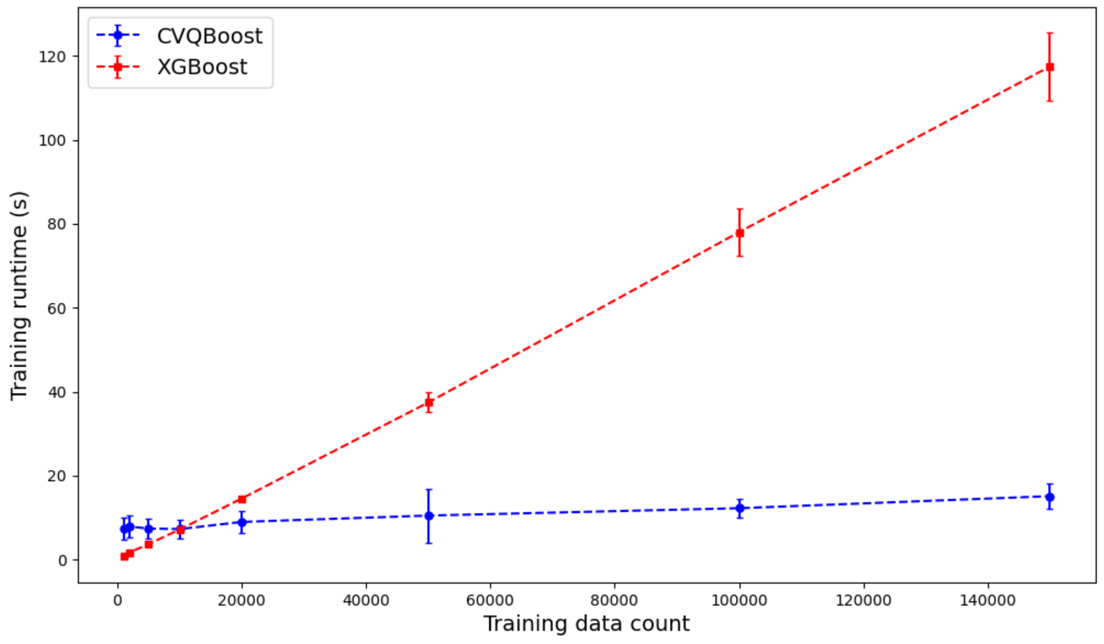


Figure 1: Training runtime of CVQBoost and XGBoost vs. count of training data samples. The 95% intervals are shown.

3.2.2 Number of Features

Table 2 summarizes the runtimes of CVQBoost and XGBoost for varying numbers of features, ranging from 5 to 38. All runtimes are reported in seconds, with each experiment repeated multiple times to ensure reliability. The table includes both the mean and standard deviation of the runtimes, as well as the breakdown of CVQBoost's runtime, where the Dirac-3 component constitutes a significant portion of the total runtime.

As the number of features increases, the runtimes of both CVQBoost and XGBoost grow. However, CVQBoost consistently demonstrates significantly shorter runtimes compared to XGBoost.

Figure 2 further illustrates the relationship between runtime and feature count for both methods. Notably, CVQBoost exhibits a more favorable runtime scaling as the feature dimension increases, making it particularly advantageous for datasets with a larger number of features.

num features	cvqboost train time	xgboost train time	dirac3 time
5	7.6 +/- 1.3	13.7 +/- 0.3	1.2 +/- 0.4
10	7.3 +/- 1.3	18.2 +/- 0.8	1.2 +/- 0.4
20	8.8 +/- 1.3	26.3 +/- 0.5	1.4 +/- 0.5
30	9.5 +/- 1.2	34.4 +/- 3.8	1.2 +/- 0.4
38	10.0 +/- 1.4	41.1 +/- 6.6	1.6 +/- 0.5

Table 2: Breakdown of CVQBoost and XGBoost runtimes for different counts of features; training data count: $50,000$.

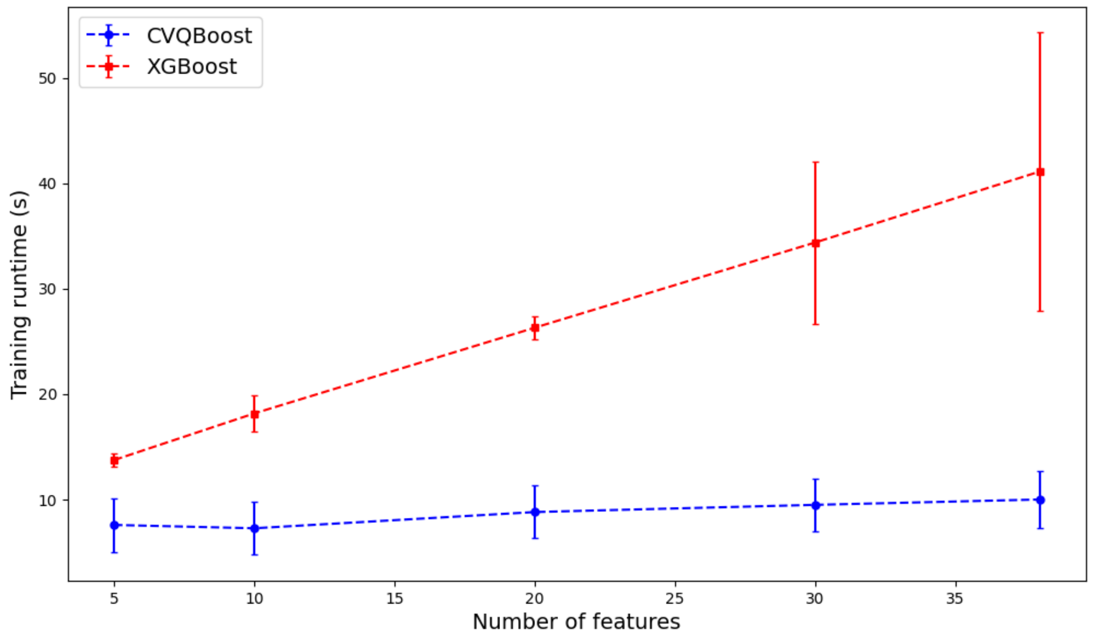


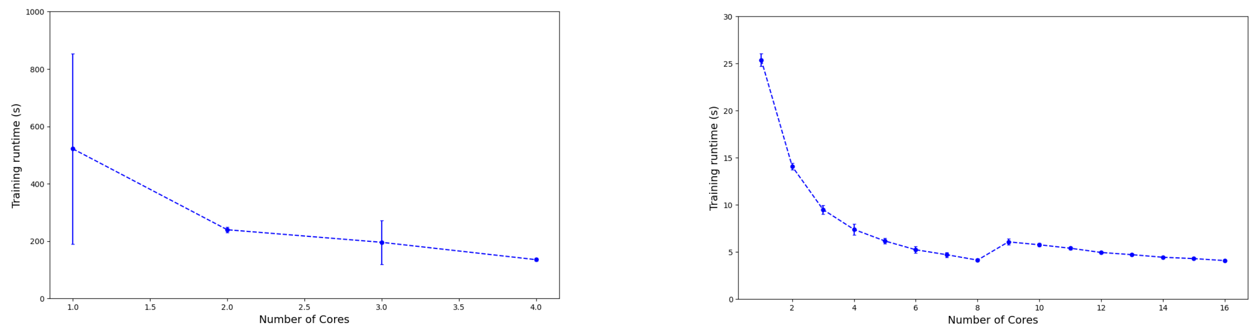
Figure 2: Training runtime of CVQBoost and XGBoost vs. number of features. The 95% intervals are shown.

3.2.3 XGBoost on Multiple Cores

The performance of XGBoost was evaluated on machines with different numbers of cores to assess its scalability in a multi-core environment. Figure 3 illustrates the training runtimes of XGBoost as a function of the number of cores utilized on an Apple M2 machine (4 cores) and a high-performance machine with 16 cores, shown in subfigures (a) and (b), respectively.

For the Apple M2 machine, as shown in Figure 3(a), the training runtime decreases initially as more cores are employed. However, this decrease plateaus as the number of cores approaches the system's maximum capacity. Similarly, on the 16-core machine (Figure 3(b)), the trend is more pronounced; while the runtime reduces with the addition of cores, the rate of improvement diminishes significantly beyond 8 cores, and further scaling provides only marginal benefits.

This saturation in scalability highlights a fundamental limitation of XGBoost in leveraging large numbers of cores for training speedup. In contrast, CVQBoost on the Dirac-3 platform exhibits remarkable scalability. By leveraging quantum computing's inherent parallelism, CVQBoost maintains consistent runtime performance improvements as dataset size and feature count grow, as discussed in earlier sections. Unlike XGBoost, which faces hardware limitations in multi-core environments, CVQBoost's design allows it to harness the power of quantum hardware effectively, avoiding the saturation observed with classical multi-core scaling.



(a) Apple M2 machine (4 cores) (b) High-performance machine (16 cores)

Figure 3: Training runtime of XGBoost vs. number of cores. (a) Apple M2 machine with 4 cores; (b) High-performance machine with 16 cores. The 95% intervals are shown; training data count: 150,000, number of features: 38.

As shown in Figure 3(b), using 8 cores for XGBoost appears to be optimal for minimizing training runtime.

Building on the scalability experiments, we conducted further tests to compare the performance of CVQBoost and XGBoost running in parallel on 8 cores, focusing on varying dataset sizes. As shown in Table 3 and Figure 4, XGBoost achieves shorter training runtimes for smaller datasets due to its efficient parallelization. While the slopes of the CVQBoost and parallelized XGBoost runtimes are similar for these relatively small datasets (ranging from 1,000 to 150,000 samples), experiments with much larger datasets reveal that CVQBoost scales significantly better with dataset size. This will be discussed further in Section 4.

train data count	xgboost total train time
1000	0.4 +/- 0.1
2000	0.5 +/- 0.0
5000	0.6 +/- 0.0
10000	0.8 +/- 0.0
20000	1.0 +/- 0.0
50000	1.8 +/- 0.0
100000	3.0 +/- 0.0
150000	4.2 +/- 0.1

Table 3: Parallelized XGBoost runtimes on 8 cores for different training data counts; number of features: 38.

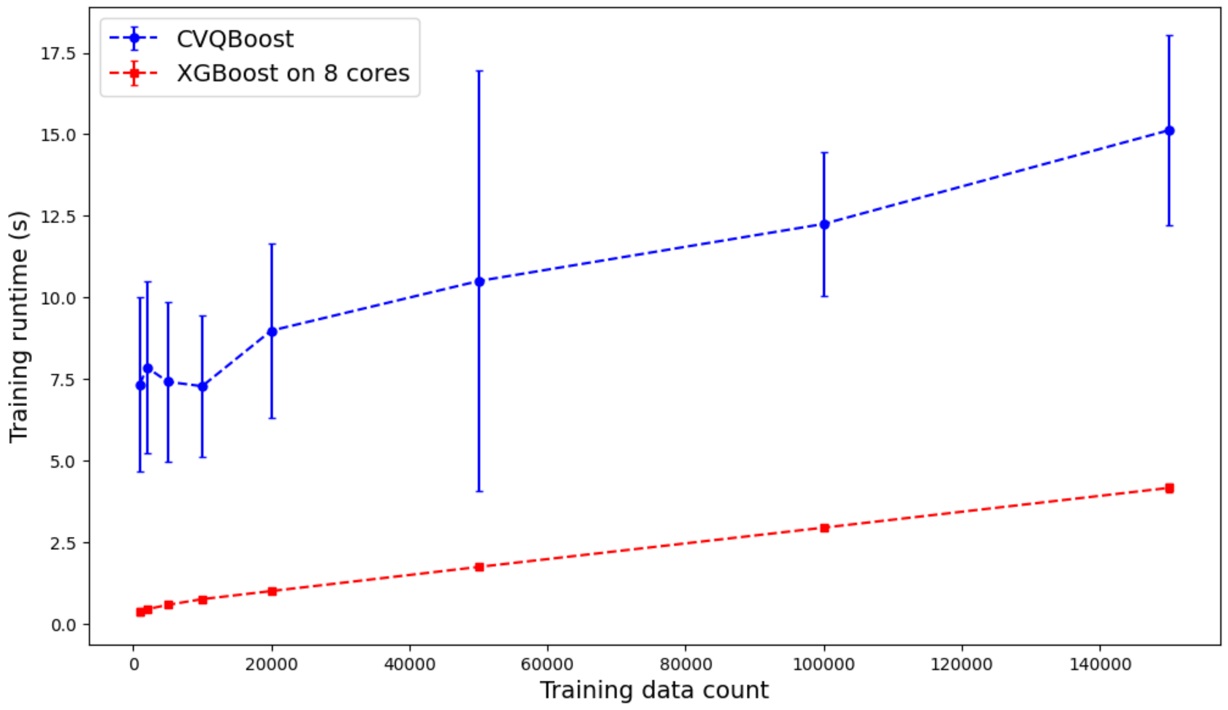


Figure 4: Training runtime of CVQBoost and XGBoost run on 8 cores in parallel.

3.2.4 XGBoost on GPU

To evaluate the performance of XGBoost in a GPU-accelerated environment, we ran the algorithm on an NVIDIA L4 GPU for various training dataset sizes, as shown in Table 4. The NVIDIA RAPIDS AI package, version 24.12.00, was used. The GPU implementation demonstrates reduced runtimes compared to CVQBoost, particularly for smaller datasets. This efficiency arises from the GPU's ability to parallelize computations, which aligns well with the nature of XGBoost's gradient boosting framework.

While CVQBoost and XGBoost exhibit similar scalability on the GPU for the relatively small dataset sizes used here (ranging from 1,000 to 150,000 samples), as shown in Figure 5, further investigation reveals that CVQBoost scales significantly better on much larger datasets. This is discussed in Section 4.

train data count	xgboost total train time
1000	1.3 +/- 0.1
2000	1.4 +/- 0.0
5000	1.5 +/- 0.0
10000	1.7 +/- 0.0
20000	1.9 +/- 0.0
50000	2.3 +/- 0.0
100000	3.0 +/- 0.0
150000	3.6 +/- 0.0

Table 4: XGBoost runtimes on GPU for different training data counts; number of features: 38.

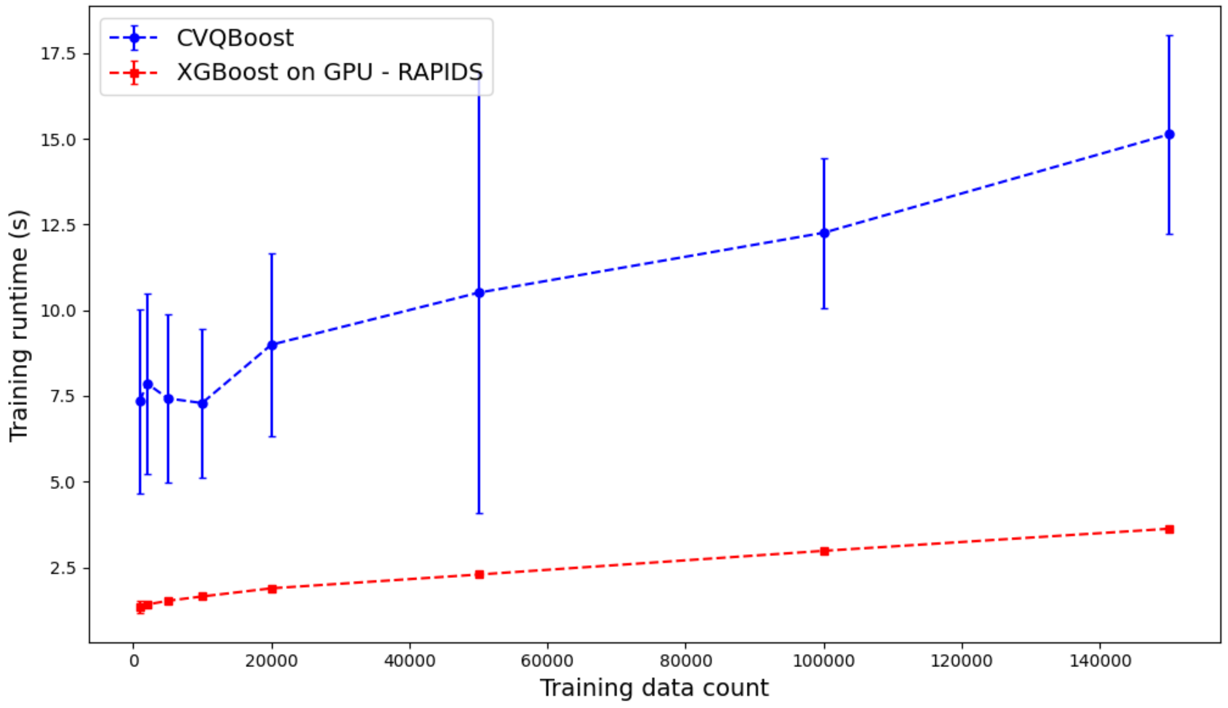


Figure 5: Comparison of training runtimes for CVQBoost and XGBoost on GPU using the NVIDIA RAPIDS AI package.

3.3 Accuracy Profiling

Handling class imbalance is a central challenge in machine learning, particularly in applications such as fraud detection, where the minority class represents rare events. In the original dataset used for this study, fraud cases (the minority class) constitute less than 0.1% of the data. To address this extreme imbalance, four widely-used data balancing strategies—ADASYN, SMOTE, SMOTE-SVM, and majority class downsampling—were applied to the training data. The test data remained unchanged to ensure an unbiased evaluation of model performance. The accuracy of CVQBoost and XGBoost was assessed across six levels of class imbalance, characterized by minority-to-majority class ratios of 0.01, 0.02, 0.05, 0.1, 0.5, and 1.0. The Area Under the Curve (AUC), calculated on the test data, was used to measure model accuracy. Table 5 provides a detailed comparison of AUC values across all experiments, while Figure 6 illustrates the trends for each balancing strategy.

The results reveal interesting patterns that provide insight into the behavior of the two algorithms under varying levels of class imbalance. When the balancing is minimal—that is, when the minority-to-majority class ratio remains small (0.01 or 0.02)—XGBoost achieves higher AUC scores compared to CVQBoost. This suggests that XGBoost is better able to leverage the heavily skewed class distribution in the early stages of balancing. However, as the class ratio increases and the training data becomes more balanced, CVQBoost's performance improves steadily and catches up to that of XGBoost, eventually achieving comparable AUC scores.

The effect of balancing strategies is particularly noticeable when using ADASYN. ADASYN generates synthetic samples by focusing on regions of the feature space that are prone to misclassification, thereby improving the representation of the minority class. As the minority-to-majority class ratio increases, CVQBoost not only matches but eventually surpasses XGBoost in terms of AUC. This can be seen clearly in both the table and Figure 6(a), where CVQBoost achieves a slight but consistent improvement over XGBoost at higher class ratios. This suggests that CVQBoost is particularly effective at learning from the diversity introduced by ADASYN-generated synthetic samples.

A similar trend is observed with SMOTE and SMOTE-SVM, where both algorithms benefit as the class ratio increases. However, while XGBoost maintains a slight edge at smaller ratios, CVQBoost consistently narrows the gap as the balancing becomes more pronounced. The superior performance of CVQBoost at higher class ratios reflects its ability to generalize better when sufficient representation of the minority class is provided.

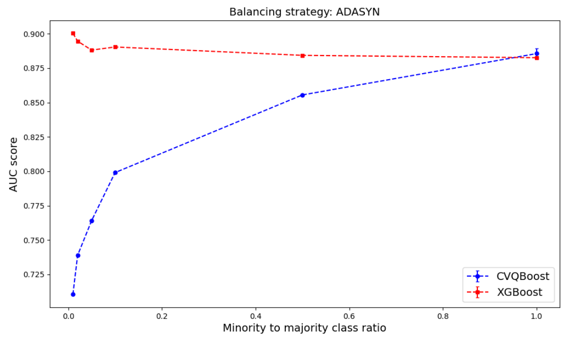
In the case of majority class downsampling, where no synthetic data is introduced and the majority class is simply reduced in size, both algorithms demonstrate gradual improvements in AUC as the class ratio increases. However, the rate of improvement is less pronounced compared to the oversampling methods. Even in this scenario, CVQBoost matches and, at times, exceeds the performance of XGBoost as the training data becomes more balanced.

Overall, these results indicate that XGBoost is initially more effective at learning from highly imbalanced data with minimal balancing, as reflected by its superior AUC scores at low class ratios.

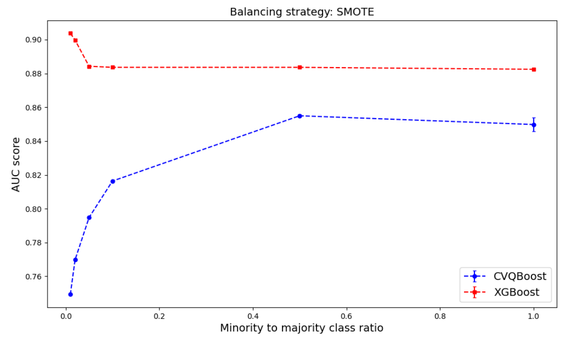
However, as the minority-to-majority class ratio increases, CVQBoost catches up and often surpasses XGBoost, particularly when using ADASYN. This highlights the adaptability and robustness of CVQBoost, especially in scenarios where synthetic balancing methods enhance the representation of the minority class. Table 5 and Figure 6 collectively demonstrate that CVQBoost becomes increasingly advantageous as class imbalance is mitigated, offering superior performance for moderately to highly balanced training data.

sampling strategy	minority to majority class ratio	cvqboost auc	xgboost auc
ADASYN	0.01	0.7105 +/- 0.0	0.9003 +/- 0.0
ADASYN	0.02	0.7388 +/- 0.0	0.8946 +/- 0.0
ADASYN	0.05	0.7642 +/- 0.0	0.8881 +/- 0.0
ADASYN	0.1	0.799 +/- 0.0	0.8904 +/- 0.0
ADASYN	0.5	0.8555 +/- 0.0002	0.8843 +/- 0.0
ADASYN	1.0	0.8855 +/- 0.0019	0.8826 +/- 0.0
SMOTE	0.01	0.7493 +/- 0.0	0.9037 +/- 0.0
SMOTE	0.02	0.7697 +/- 0.0	0.8998 +/- 0.0
SMOTE	0.05	0.7948 +/- 0.0	0.8842 +/- 0.0
SMOTE	0.1	0.8164 +/- 0.0	0.8836 +/- 0.0
SMOTE	0.5	0.855 +/- 0.0004	0.8836 +/- 0.0
SMOTE	1.0	0.8498 +/- 0.0021	0.8824 +/- 0.0
SVMSMOTE	0.01	0.7674 +/- 0.0	0.9058 +/- 0.0
SVMSMOTE	0.02	0.7902 +/- 0.0	0.9019 +/- 0.0
SVMSMOTE	0.05	0.8092 +/- 0.0	0.9027 +/- 0.0
SVMSMOTE	0.1	0.8263 +/- 0.0	0.9051 +/- 0.0
SVMSMOTE	0.5	0.8582 +/- 0.0014	0.9033 +/- 0.0
SVMSMOTE	1.0	0.8569 +/- 0.001	0.902 +/- 0.0
Downsampling	0.01	0.7415 +/- 0.0	0.9105 +/- 0.0
Downsampling	0.02	0.7698 +/- 0.0	0.9056 +/- 0.0
Downsampling	0.05	0.7948 +/- 0.0	0.9079 +/- 0.0
Downsampling	0.1	0.8167 +/- 0.0	0.9007 +/- 0.0
Downsampling	0.5	0.8689 +/- 0.0004	0.8915 +/- 0.0
Downsampling	1.0	0.8508 +/- 0.0016	0.8872 +/- 0.0

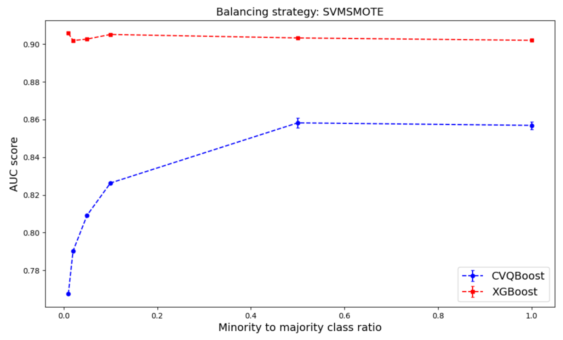
Table 5: AUC scores for CVQBoost and XGBoost using different balancing strategies and minority to majority class ratios.



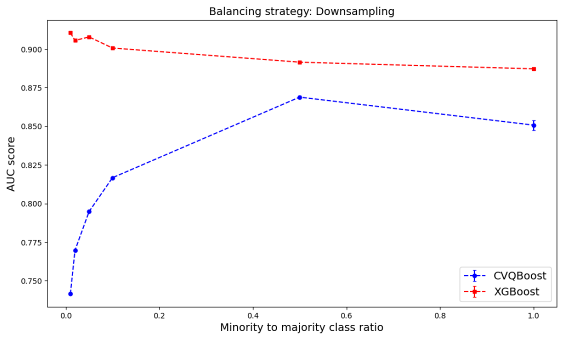
(a) ADASYN



(b) SMOTE



(c) SMOTE-SVM



(d) Downsampling

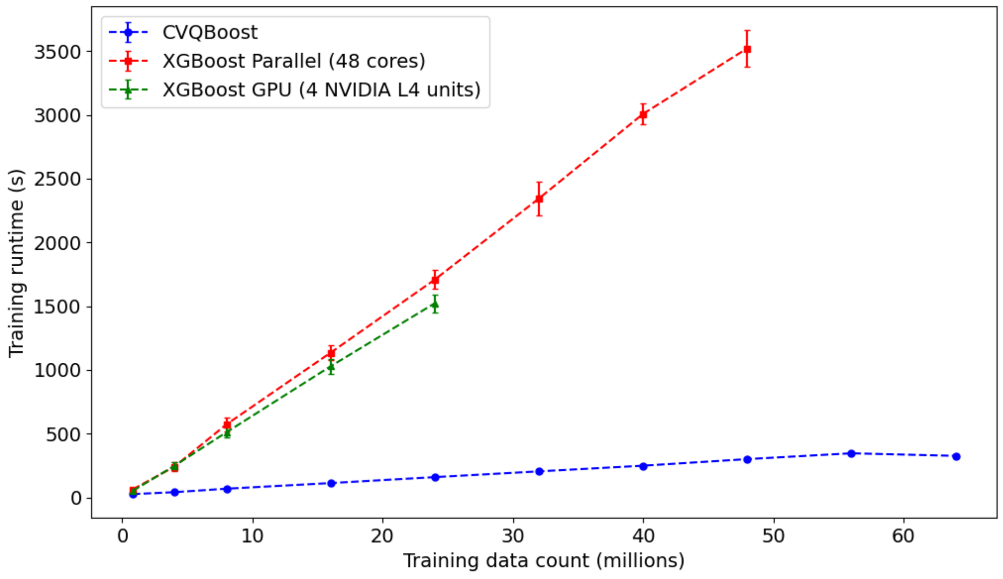
Figure 6: AUC score of CVQBoost vs. minority-to-majority class ratio in training datasets using different balancing strategies: (a) ADASYN, (b) SMOTE, (c) SMOTE-SVM, and (d) majority class downsampling. The 95% intervals are shown.

4 Scalability on Large-Scale Synthetic Data

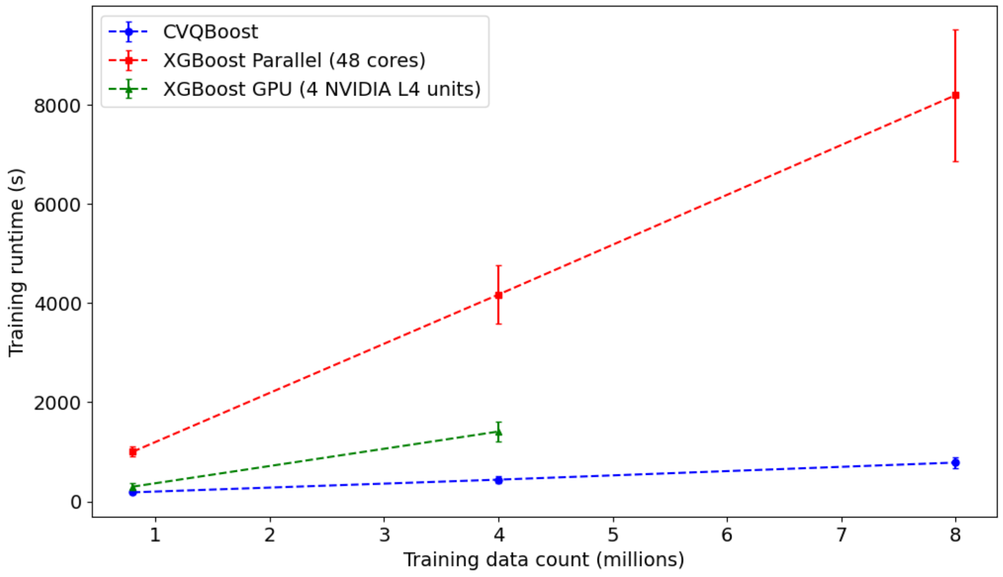
To further evaluate the scalability of CVQBoost, we generated large-scale synthetic datasets using the Scikit-learn package. The dataset sizes range from 1M to 70M samples, with 80% allocated for training. Each dataset consists of 100 features.

Experiments were conducted on a high-performance computing system equipped with 48 CPU cores (2.64 GHz) and four NVIDIA L4 GPUs. Each GPU unit has 24GB of memory, totaling 96GB across all four units. XGBoost was tested under two configurations: (1) utilizing 48 CPU cores and (2) running on four NVIDIA L4 GPUs via the NVIDIA RAPIDS AI framework.

These results highlight the scalability advantage of CVQBoost for large-scale classification tasks, where computational efficiency is critical. Figure 8 further demonstrates that CVQBoost maintains linear scalability with increasing feature dimensions, whereas XGBoost exhibits quadratic runtime growth. Additionally, Figures 7(a) and 7(b) illustrate CVQBoost's memory efficiency, as it is able to process significantly larger datasets than XGBoost on both CPU and GPU, which are constrained by memory limitations.



(a) 100 features



(b) 500 features

Figure 7: Training runtime of CVQBoost vs. XGBoost on 48 CPUs and four NVIDIA L4 GPUs. Error bars indicate 95% intervals. CVQBoost exhibits superior scalability for large datasets. In (a) (100 features), CVQBoost corresponds to dataset sizes of 1M to 70M, whereas XGBoost on 48 cores stops at 50M due to RAM limitations, and on GPU, it stops at 30M due to GPU memory constraints. In (b) (500 features), CVQBoost and XGBoost on CPU cover 1M to 10M, while XGBoost on GPU is limited to 1M to 5M due to memory restrictions.

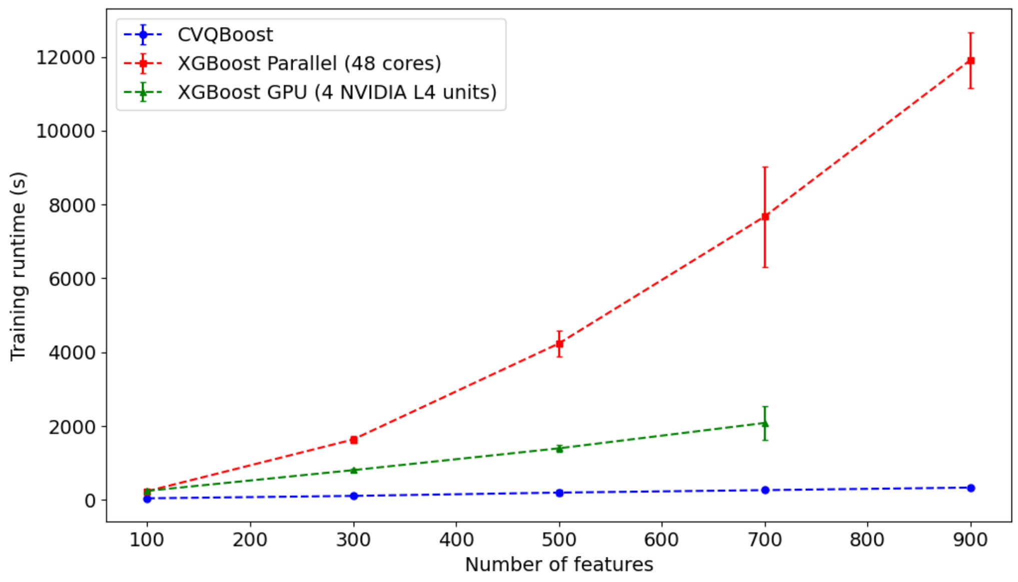


Figure 8: Training runtime of CVQBoost vs. XGBoost on 48 CPUs and four NVIDIA L4 GPUs as the number of features increases. Error bars indicate 95% intervals. CVQBoost runs faster and scales linearly with a modest slope, whereas XGBoost runtimes increase quadratically.

References

[1] Yoav FREUND, Robert SCHAPIRE, and Naoki ABE. “A Short Introduction to Boosting”. In: *Journal of the Japanese Society for Artificial Intelligence* 14.5 (1999), pp. 771–780. DOI: 10.11517/jjsai.14.5_771.

[2] Hartmut Neven et al. “QBoost: Large Scale Classifier Training withAdiabatic Quantum Optimization”. In: *Proceedings of the Asian Conference on Machine Learning*. Ed. by Steven C. H. Hoi and Wray Buntine. Vol. 25. Proceedings of Machine Learning Research. Singapore Management University, Singapore: PMLR, Apr. 2012, pp. 333–348. URL: https://proceedings.mlr.press/v25/neven12.html.

[3] MACHINE LEARNING GROUP – ULB. *Credit Card Fraud Detection*. URL: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud.