Predicting Major League Baseball Playoffs Using Quantum Machine Learning

1st Danesh Morales Hashemi
MMATH Student - Applied Mathematics
University of Waterloo
Waterloo, ON
d2morale@uwaterloo.ca

2nd Jonathan Zhu
MENG Student - Electrical and Computer Engineering
University of Waterloo
Waterloo, ON
j336zhu@uwaterloo.ca

***Abstract*—This project explores the application of Quantum Machine Learning (QML) to predict the outcomes of Major League Baseball (MLB) games, specifically the Postseason matches that took place throughout the month of October. Using features such as team batting averages, pitching performance, and game context (e.g. home vs. away), we implemented two QML algorithms: the Quantum Support Vector Machine (QSVM) and the Variational Quantum Classifier (VQC), using the Qiskit framework. We then compared these quantum classifiers to their classical counterparts, namely the Support Vector Machine (SVM) and a classical neural network, the Multi-Layer Perceptron (MLP) Classifier. We found that the QML algorithms achieved prediction accuracies comparable to those of the classical algorithms; however, training the QML models was significantly more time-consuming. We therefore conclude that, in this setting, there is no quantum advantage, and that classical prediction models remain the preferable choice.**

Contents

I. Introduction

According to American sportswriter Jeff Passan, baseball is the most over-analyzed sport in the world [1]. Every organization in Major League Baseball (MLB) has a team of data scientists who run prediction models that help the team prepare against the opponents they are facing, both before and during the game [2]. There are hundreds of metrics that can be utilized to predict the outcome of a specific matchup between two teams, and they are widely available to the public thanks to the official MLB API [3]. Combined with the use of machine learning algorithms, baseball fans all around the world can create their own prediction models as well. With the help of quantum machine learning algorithms, namely the Quantum Support Vector Machine (QSVM) and Variational Quantum Classifier (VQC), we will develop a prediction model that determines the winner of a matchup between two baseball teams. More tangentially, the main objective of this project is to predict the winner of the 2025 MLB Playoffs [4], which began on September 30 and concluded on November 1. Since the playoffs have ended, we objectively measured the success of our prediction models for both QSVM and VQC by directly comparing the actual results with the predicted results from the two QML algorithms.

II. Classical Data

A. Choosing Features

We began by carefully choosing useful metrics that could help our algorithms classify whether a team wins or loses. The main criterion for selecting features was that they should not be directly determined by the game outcome (for example, the number of runs scored was not chosen, since the team that scores the most runs always wins). We initially considered 17 candidate features; however, we observed that some of them were clearly correlated with each other. For instance, On-Base Percentage (OBP) measures the percentage of plate appearances in which a batter reaches base, while Batting Average (AVG) measures the rate of at-bats that result in a hit. Because batters typically reach base by getting a hit, and only rarely reach base without a hit, including both features was redundant. We therefore chose to keep OBP, as it also accounts for ways of getting on base without a hit. Moreover, the main motivation for reducing the number of features was to decrease the computational overhead of the QML algorithms. After a careful consideration, we ended up with 9 features.

1) Home runs (HR)
2) Hits (H)
3) On-Base Percentage (OBP)
4) Slugging (SLG)
5) Runners Left on Base (LOB)
6) Strikeouts (SO)
7) Walks and Hits per Inning Pitched (WHIP)
8) Throwing Strike Percentage (STRIKE%)
9) Starting Pitcher Earned Runs Average (SP ERA)

The definition of these terminologies can be found in the MLB Glossary [5].

B. Training Data

The data used to train our ML classifiers consisted of every game played during the 2025 Regular Season. Each team plays 162 games per year, 81 at home and 81 away. Since there are 30 teams in MLB, this results in $162 \times 30/2 = 2430$ games in total. Therefore, our training dataset contained 2430 rows, with 18 feature columns (9 for each team, home and away), and a label indicating whether the home team won (True or False). The features for each row were taken from what happened in the corresponding game; hence, we consider our training data to be **post-game** data.

C. Testing Data

The testing data consisted of the 47 playoff games played during the 2025 Postseason. Since the goal of the project was to *predict* the winner of each matchup rather than *classify* the winner based on post-game data, we populated the feature columns using each team's season average statistics, for both home and away games, instead of game-specific outcomes.

D. Data preprocessing

We reduced the number of feature columns from 18 to 9 by replacing the separate home and away features with their differences (home minus away).

E. Look-ahead Bias

There is a fundamental issue with the way we initially defined the training and testing data. The training data consists of **post-game** data, whereas the testing data is composed of **pre-game** data. This mismatch is a common mistake in machine learning, known as *look-ahead bias* [6]. To address this, we repopulated the data for each game using the average statistics from each team's last 40 games (home and away). As a consequence, the data from the first 80 games of the season for each team became unusable, effectively reducing our dataset by approximately 50%.

Completely discarding half of the training data did not seem feasible. Instead, we began counting games only after each team had played at least 10 home and 10 away games. Initially, the averages were computed over the first 10 games of the season (home and away), and as the season progressed, the sample size for the averaged data reached 40 games. Once teams reached this 40-game threshold, we used only the most recent 40 games to populate the rows of our training set. Using this method, the dataset was reduced to 2084 rows, retaining 85.8% of the original data. The size of the testing set (i.e., the playoff games) did not change; nevertheless, its rows were populated using each team's last 40 home and away games of the regular season. Thus, both the training and testing data are based on pre-game information, successfully eliminating look-ahead bias.

III. Support Vector Machine (SVM)

A. Classical SVM

IBM defines SVMs as "a supervised machine learning algorithm that classifies data by finding an optimal line or hyperplane that maximizes the distance between each class in an N-dimensional space" [7]. In this project, we will be using SVM to maximize the distance between games in a 9-dimensional space. The classical version of support vector machine served as a baseline model for computation time and accuracy. Four kernels were to identify the kernel with highest accuracy: the linear kernel, polynomial kernel, radial basis function (RBF) kernel, and sigmoid kernel. The features were standardized using the `fit_transform` function from `scikit-learn`'s `StandardScaler` class.

The standard score of a sample x from `StandardScaler` is calculated as:

$$z = \frac{x - u}{s},$$

where u is the mean of the training samples and s is the standard deviation of the training samples [8].

TABLE I
Classical Support Vector Machine Results

Metric	**Linear**	**Poly**	**RBF**	**Sigm.**
Training Time	**0.1155 s**	0.1245 s	0.1237 s	**0.1522 s**
Accuracy	55.32%	55.32%	**68.09%**	**51.06%**

As shown in Table I, the RBF kernel achieves the highest accuracy among the classical SVM methods. This accuracy and training time will be used to compare with the quantum counterparts.

B. Quantum Support Vector Classifier

One of the quantum models used was the quantum support vector classifier (QSVC). We applied the same `StandardScaler` [8] preprocessing described in Section III-A to normalize the input features prior to encoding them into quantum states. A quantum kernel is used to compute our model from a quantum circuit as apposed to a predefined kernel in the classical version. MLB data is encoded into a (simulated) quantum state through a feature map, and similarity is then measured by the fidelity between the corresponding quantum states. A quantum kernel matrix is built utilizing the feature map and fidelity.

Feature Map: The input features were encoded into a quantum circuit using the **ZZFeatureMap** and **PauliFeatureMap (Z, Y, ZZ)**. The configuration conbinations used were the following:

- Feature dimension = 9
- Number of repeat circuits = 1 & 2
- Linear entanglement

Fidelity: A fidelity-based quantum kernel was used to compute the similarity between data points. The configuration used was the following:

- ComputeUncompute fidelity estimator from `qiskit_machine_learning.kernels`
- Sampler primitive from `qiskit.primitives`
- FidelityQuantumKernel from `qiskit_machine_learning.kernels`

Sample Size: Training was performed on different sample sizes. Due to the cost of constructing and storing fidelity-based quantum kernels, QSVM exhibits quadratic scaling with respect to both the number of samples and feature dimension [9]. Because of this drawback, all samples sized above 500 samples resulted in an *out of memory* error. So, the algorithm was trained on 500 samples from the tail of the dataset. The samples were taken from the tail to train on more recent data.

TABLE II
QSVC Results for Different Feature Maps

Map (reps)	**Pauli (2)**	**Pauli (1)**	**ZZ (2)**	**ZZ (1)**
Training Time	**32.94**	20.68 min	23.30 min	**15.07 min**
Accuracy	**61.70%**	**53.19%**	55.32%	55.32%

We can observe from Table II that the Pauli feature map with two repetitions yielded the highest precision with 61.70% accuracy. However, this comes at the cost of long training time with a 32.94 minute training time.

IV. Neural Network and Quantum Classifiers

A. Multilayer Perceptron Classifier (MLPC)

he Multi-Layer Perceptron (MLP) classifier is a standard feed-forward classical neural network composed of fully connected layers that map an input feature vector (e.g., classical data) to a label or outcome [10]. It consists of an input layer, one or more hidden layers, and an output layer that produces class probabilities, with the predicted class being the one with the highest probability. The hidden layers apply nonlinear transformations, which enable the network to learn non-linear decision boundaries in the feature space.

As a baseline for benchmarking the Variational Quantum Classifier, we trained several MLP classifiers with varying parameters using `scikit-learn` [11]. The following table summarizes the performance of the five best-performing MLP classifiers:

Table III summarizes the performance of the best MLP classifiers over the playoff games. The highest accuracy (61.7%)

TABLE III
MLPC Results with different parameters

Optimizer	**Layers**	α	**Max Iter**	**Accuracy**	**Time**
Adam	(3,)	0.0010	300	**61.7%**	0.349 s
Adam	(3, 3)	0.0010	300	59.67%	0.241 s
L-BFGS	(5, 5)	0.0001	300	59.67%	**0.732 s**
Adam	(5,)	0.0010	300	**57.47%**	**0.237 s**
SGD	(5,)	0.0001	500	**57.47%**	0.243 s
L-BFGS	(3,)	0.0001	300	**57.47%**	0.585 s

was obtained by the Adam optimizer with 3 hidden layers. Interestingly, increasing the number of hidden layers did not lead to an improvement in the accuracy. Overall, the training time of the MLPC classifiers was under 1 second.

B. Variational Quantum Classifier (VQC)

A variational quantum classifier (VQC) is a QML model built from a variational quantum circuit whose parametrized gates are optimized with the assistance of a classical optimizer to perform data classification [12].

Similar to the QSVM, a VQC first maps classical data into a Hilbert space using a feature map. Once the data is encoded into quantum states, we apply a parameterized quantum circuit, also called *ansatz*, which depends on a vector of trainable parameters $\vec{\theta}$ [12]. We then measure the circuit's output and use these measurement results to compute a cost function. A classical optimizer updates $\vec{\theta}$ to minimize this cost, creating a feedback loop of "run circuit $\rightarrow$ measure $\rightarrow$ update $\vec{\theta}$" that continues until the cost function converges to a minimum and the classifier is properly trained [13].

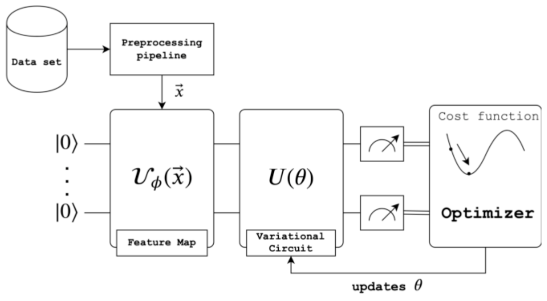

Fig. 1. Diagram of the VQC protocol [14]

We exhaustively trained 48 VQCs using Qiskit's `VQC` class [15], exploring different combinations of feature maps, ansatz, and optimizers. The breakdown is as follows:

1) **Feature Maps:**
   a) ZFeatureMap (1-2 repetitions)
   b) ZZFeatureMap (1-2 repetitions)
2) **Ansatz:**
   a) TwoLocal (1-2 repetitions)
   b) EfficientSU2 (1-2 repetitions)
   c) RealAmplitudes (1-2 repetitions)
3) **Classical Optimizers:**
   a) SPSA (max 500 iterations)
   b) COBYLA (max 300 iterations)

TABLE IV
VQC RESULTS FOR DIFFERENT CONFIGURATIONS

Feature Map (Reps)	Ansatz (Reps)	Optimizer	Accuracy
Z (2)	TwoLocal (1)	SPSA500	**65.96**%
ZZ (1)	EffSU2 (1)	COBYLA300	61.70%
Z (2)	EffSU2 (2)	COBYLA300	61.70%
ZZ (2)	RealAmps (1)	SPSA500	59.57%
Z (2)	TwoLocal (2)	SPSA500	59.57%

Table IV displays the accuracy of the five best-performing VQCs. The highest accuracy, 65.96%, is obtained with the `ZFeatureMap` (two repetitions), a `TwoLocal` ansatz with one repetition, and the SPSA optimizer (500 iterations). The training time for this VQC was 2 hours and 42 minutes. Tied for fourth place (59.57%) is the same configuration, except with two ansatz repetitions instead of one; increasing the training time to 3 hours and 7 minutes; however, it yields a lower accuracy. Similar to the MLPC case, using more resources does not necessarily lead to better performance.

V. RESULTS

We achieved an accuracy of 61.7% and 66.0% with our best-performing QSVM and VQC quantum machine learning algorithms The following is a detailed table of all the matchups and predictions. Table V provides an in-depth breakdown of the predictions for all 47 playoff games made by the QSVM and VQC algorithms.

VI. COMPARING CLASSICAL VS QUANTUM ALGORITHMS

The following is a breakdown of the accuracy for each (best tested) algorithm for each series in the 2025 MLB playoffs.

We see in Table VI that for the classical version of Support Vector Machine has a higher accuracy when compared to its quantum counterpart. Conversely, the quantum version of Variational Quantum Classifier scored a higher accuracy when compared to its classical counterpart. When comparing the results of all tested algorithms, we find that the classical Support Vector Machine scores the highest overall accuracy of 68.1% while the Variational Quantum Classifier comes in at a close second with 66.0% accuracy.

VII. CONCLUSION

In this project, we investigated how quantum machine learning methods, specifically QSVM and VQC, compare to classical machine learning methods when predicting MLB postseason outcomes. By constructing a consistent pre-game and post-game dataset, we eliminate look-ahead bias to train and test classical and quantum machine learning models. We found that the quantum machine learning models achieved comparable accuracy to their classical baselines, but require substantially greater computational resources and training time. While the Variational Quantum Classifier performed competitively, even surpassing its classical counterpart, the classical Support Vector Machine using the Radial Basis Function kernel ultimately delivered the highest overall accuracy with significantly less training time. These results suggest that quantum models do not yet offer a practical advantage for baseball prediction. Although, the promising performance of quantum models tested indicate that quantum machine learning remains promising for the future of sports analytics as larger quantum hardware and more expressive feature maps continue to advance.

TABLE V
COMPARISON OF ACTUAL WINNERS AND MODEL PREDICTIONS (QSVM VS VQC)

Series/Game	**Actual Winner**	**QSVM (Best)**	**VQC (Best)**
WS G1	TOR	**TOR**	**LAD**
WS G2	LAD	**TOR**	**LAD**
WS G3	LAD	**LAD**	**TOR**
WS G4	TOR	**LAD**	**TOR**
WS G5	TOR	**LAD**	**TOR**
WS G6	LAD	**TOR**	**LAD**
WS G7	LAD	**TOR**	**LAD**
ALCS G1	SEA	**TOR**	**TOR**
ALCS G2	SEA	**TOR**	**TOR**
ALCS G3	TOR	**SEA**	**SEA**
ALCS G4	TOR	**SEA**	**SEA**
ALCS G5	SEA	**SEA**	**SEA**
ALCS G6	TOR	**TOR**	**TOR**
ALCS G7	TOR	**TOR**	**TOR**
NLCS G1	LAD	**MIL**	**LAD**
NLCS G2	LAD	**MIL**	**LAD**
NLCS G3	LAD	**LAD**	**LAD**
NLCS G4	LAD	**LAD**	**LAD**
ALDS1 G1	TOR	**TOR**	**TOR**
ALDS1 G2	TOR	**TOR**	**TOR**
ALDS1 G3	NYY	**NYY**	**TOR**
ALDS1 G4	TOR	**NYY**	**TOR**
ALDS2 G1	DET	**SEA**	**SEA**
ALDS2 G2	SEA	**SEA**	**SEA**
ALDS2 G3	SEA	**DET**	**DET**
ALDS2 G4	DET	**DET**	**DET**
ALDS2 G5	SEA	**SEA**	**SEA**
NLDS G1	MIL	**MIL**	**MIL**
NLDS G2	MIL	**MIL**	**MIL**
NLDS G3	CHC	**CHC**	**CHC**
NLDS G4	CHC	**CHC**	**CHC**
NLDS G5	MIL	**MIL**	**MIL**
NLDS G1	LAD	**LAD**	**PHI**
NLDS G2	LAD	**LAD**	**PHI**
NLDS G3	PHI	**PHI**	**PHI**
NLDS G4	LAD	**LAD**	**PHI**
ALWC G1	DET	**CLE**	**CLE**
ALWC G2	CLE	**CLE**	**CLE**
ALWC G3	DET	**CLE**	**CLE**
ALWC2 G1	BOS	**NYY**	**NYY**
ALWC2 G2	NYY	**NYY**	**NYY**
ALWC2 G3	NYY	**NYY**	**NYY**
NLWC G1	LAD	**LAD**	**LAD**
NLWC G2	LAD	**LAD**	**LAD**
NLWC2 G1	CHC	**CHC**	**CHC**
NLWC2 G2	SD	**CHC**	**CHC**
NLWC2 G3	CHC	**CHC**	**CHC**

TABLE VI
COMPARISON OF ACCURACY PER SERIES AND OVERALL

Series	**Teams**	**SVM**	**QSVM**	**MLPC**	**VQC**
World Series	TOR vs LAD	**71.4%**	**28.6%**	57.1%	**71.4%**
ALCS	SEA vs TOR	**71.4%**	**42.9%**	**42.9%**	57.1%
NLCS	LAD vs MIL	**50.0%**	**50.0%**	**50.0%**	**100.0%**
ALDS 1	TOR vs NYY	**75.0%**	**75.0%**	**75.0%**	**75.0%**
ALDS 2	DET vs SEA	**40.0%**	**60.0%**	**40.0%**	**60.0%**
NLCS 1	MIL vs CHC	**60.0%**	**100.0%**	**100.0%**	**100.0%**
NLCS 2	LAD vs PHI	**100.0%**	**100.0%**	**25.0%**	**25.0%**
ALWC 1	DET vs CLE	**66.7%**	**33.3%**	**66.7%**	**33.3%**
ALWC 2	BOS vs NYY	**16.7%**	**66.7%**	**66.7%**	**66.7%**
NLWC 1	LAD vs CIN	**100.0%**	**100.0%**	**100.0%**	**100.0%**
NLWC 2	CHC vs SD	**66.7%**	**66.7%**	**66.7%**	**66.7%**
Total Accuracy	–	**68.1%**	**61.7%**	**61.7%**	66.0%

VIII. Future Work

Future work will focus on expanding both the feature engineering and the quantum modeling components of this study. We plan to incorporate more advanced features, such as afternoon vs. evening performance splits and measures of team fatigue based on recent travel to face opposing teams. This will allow the models to learn some of the more intuitive nuances of baseball. We also plan to expand the dataset to include data from years prior to 2025 along with a larger rolling window. This will allow for larger training sets and more robust generalization analysis across different periods of time.

On the quantum side, we want to try out more combinations of feature maps, ansatz, and optimizers to see how they affect performance. This includes testing with different variations of ZZFeatureMap, and comparing them with higher-order Pauli maps or different entanglement patterns.

For Quantum Support Vector machine specifically, we plan to explore Quantum Support Vector Machines based on the original least-squares proposed by Rebentrost et al. [16]. Their approach leverages quantum linear-algebra subroutines to solve SVM optimization problems with logarithmic complexity in both the number of training samples and feature dimension. Implementing this technique could potentially allow evaluations for exponential speed-ups proposed in the literature. This could allow for larger MLB datasets to train and faster training time.

For Variational Quantum Classifier specifically, we plan to test deeper or more efficient ansatz, as well as a wider range of optimizers such as Adam and Conjugate Gradient (CG). By default, Qiskit uses a cross-entropy loss function, but we can also experiment with alternative loss functions, such as the `L1Loss` and `L2Loss` cost functions provided by Qiskit. The goal is to determine whether new combinations of ansatz, optimizer, and loss function can improve the quantum model's accuracy.

Appendix A – Versions / Environment

Classical Support Vector Machine and Quantum Support Vector Machine were tested and computed using Jupyter Notebook on the following hardware: Windows 11 PC, Intel i7-9700K (8 Cores), NVIDIA RTX 2070 SUPER, 32 GB RAM. This machine tested on the following import versions:

- **python**: 3.12.0
- **pandas**: 2.3.2
- **scikit-learn**: 1.7.2
- **qiskit**: 1.4.5
- **qiskit_machine_learning**: 0.8.4
- **qiskit_algorithms**: 0.3.1

Multi-core processing and GPU acceleration were not enabled.

Multilayer Perceptron Classifier and Variational Quantum Classifier were tested and computed using Jupyter Notebook on the following hardware: Windows 11 PC, Intel i9-13900K (24 Cores), NVIDIA RTX 4090, 64 GB RAM. This machine tested on the following import versions:

- **python**: 3.13.7
- **pandas**: 2.3.2
- **scikit-learn**: 1.7.2
- **qiskit**: 1.4.5
- **qiskit_machine_learning**: 0.8.4
- **qiskit_algorithms**: 0.4.0

Multi-core processing and GPU acceleration were not enabled.

References

[1] Jeff Passan. *The Arm: Inside the Billion-Dollar Mystery of the Most Valuable Commodity in Sports*. HarperCollins, 2016.

[2] Claire Newman. Beyond moneyball: Phillies data scientist give students a real-world look at how today's mlb teams use data. https://datascience.virginia.edu/news/beyond-moneyball-phillies-data-scientist-give-students-real-world-look-how-todays-mlb-teams, 5 2023.

[3] python-mlb-statsapi developers. python-mlb-statsapi: Unofficial python wrapper for the mlb stats api. Python package on PyPI, 2025. Accessed 2025-12-01.

[4] MLB Advanced Media, LP. Mlb postseason 2025: Playoff bracket & schedule. https://www.mlb.com/postseason, 2025.

[5] Major League Baseball. Mlb glossary. https://www.mlb.com/glossary, 2025. Accessed 2025-12-01.

[6] The Anh Pham. Look-ahead bias in quantitative finance: The silent killer of trading strategies. *Medium*, 8 2024. Published in Funny AI & Quant.

[7] E. Kavlakoglu. Support-vector machine (svm). https://www.ibm.com/think/topics/support-vector-machine. [Online; accessed 1-Dec-2025].

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] Gabriela Pinheiro, Donovan Slabbert, Luis Kowada, and Francesco Petruccione. Quantum kernel and hhl-based support vector machines for multi-class classification, 2025.

[10] Ginni Garg, Dheeraj Kumar, ArvinderPal, Yash Sonker, and Ritu Garg. A hybrid MLP–SVM model for classification using spatial-spectral features on hyper-spectral images. *arXiv preprint arXiv:2101.00214*, 2021.

[11] Scikit-Learn. *MLPClassifier*. scikit-learn, 2025. Accessed 2025-12-01.

[12] IBM Quantum. Quantum variational circuits and quantum neural networks. IBM Quantum Learning, Quantum Machine Learning course, 2025. Lesson: QVCs and QNNs. Accessed 2025-12-01.

[13] Qiskit Machine Learning Developers. Training a quantum model on a real dataset. Qiskit Machine Learning Tutorials, 2025. Accessed 2025-12-01.

[14] Qmunity. Building a variational quantum classifier. *Q-munity, The Quantum Insider*, 6 2024. Accessed 2025-12-01.

[15] Qiskit Machine Learning Developers. *VQC - Qiskit Machine Learning 0.8.4*. Qiskit, 2025. API reference for the Variational Quantum Classifier (VQC). Accessed 2025-12-01.

[16] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical Review Letters*, 113(13), September 2014.