5-2023

Quantum Artificial Intelligence Supported Autonomous Truck Platooning

Pronab Kumar Biswas
pronabb@g.clemson.edu

QUANTUM ARTIFICIAL INTELLIGENCE SUPPORTED AUTONOMOUS TRUCK PLATOONING

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
Of the Requirements for the Degree
Master of Science
Civil Engineering

---

by
Pronab Kumar Biswas
May 2023

Accepted by:
Dr. Mashrur Chowdhury, Committee Chair
Dr. Sakib Mahmud Khan
Dr. Yao Wang

ABSTRACT

Truck platooning can potentially increase the operational efficiency of freight movement on U.S. corridors, improving commercial productivity and economic vibrancy. Predicting each leader vehicle trajectory in the autonomous truck platoon using Artificial Intelligence (AI) can enhance platoon efficiency and safety. Reliance on classical AI may not be efficient for this purpose as it will increase the computational burden for each truck in the platoon. However, Quantum Artificial Intelligence (AI) can be used in this scenario to enhance learning efficiency, learning capacity, and run-time improvements. This study developed and evaluated a Long Short-Term Memory Networks (LSTM) model and a hybrid quantum-classical LSTM (QLSTM) for predicting the trajectory of each leader vehicle of an autonomous truck platoon. Both the LSTM and QLSTM provided comparable results. However, Quantum-AI is more efficient in real-time management for an automated truck platoon as it requires less computational burden. The QLSTM training required less data compared to LSTM. Moreover, QLSTM also used fewer parameters compared to classical LSTM. This study also evaluated an autonomous truck platoon's operational efficacy and string stability with the prediction of trajectory from both classical LSTM and QLSTM using the Intelligent Driver Model (IDM). The platoon operating with LSTM and QLSTM trajectory prediction showed comparable operational efficiency. Moreover, the platoon operating with QLSTM trajectory prediction provided better string stability compared to LSTM.

DEDICATION

To my beloved parents, my wife, and my daughter for whom my whole life pertains.

ACKNOWLEDGEMENTS

I would like to express my gratitude to everyone who played a part in helping me successfully graduate from the Glenn Department of Civil Engineering at Clemson University. Firstly, I want to thank my advisor, Dr. Mashrur Chowdhury, for his guidance and support during my graduate studies. I feel fortunate to have known him and to have been a part of his research group. Secondly, I am grateful to Dr. Sakib Mahmud Khan and Dr. Yao Wang for their insightful comments and suggestions, which helped me to refine my thesis. Additionally, I want to thank the members of the research group for their valuable feedback and scholarly discussions. I would like to give special thanks to Sabbir Salek for his assistance with the research. I am also thankful to my friends at Clemson, who made Clemson my second home, thousands of miles away from my family. A particular thank you goes to Kristin Baker, who helped me with administrative tasks since my first day as a graduate student in the Glenn Department of Civil Engineering at Clemson University. Above all, I would like to express my deepest gratitude to Methila Sarker Pooja, my better half, who is the reason for my achievements so far in life and makes me who I am.

TABLE OF CONTENTS

Page

Table of Contents (Continued) Page

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER ONE

INTRODUCTION

1.1 Background and Motivation

The joining of two or more trucks in convoy utilizing wireless connectivity and autonomous driving assistance systems is known as truck platooning. When these vehicles are connected for some distance on a highway, they automatically maintain a safe, close distance between each other. According to the U.S. Department of Transportation, 72 percent of goods in the U.S. are transported by trucks; therefore, finding safer and more efficient ways to move them is essential (Economics and Industry Data, 2022). Truck platooning can potentially increase the operational efficiency of freight movements on U.S. corridors to improve commercial productivity and economic vibrancy. Although real-world deployments of truck platooning are still in their infancy, a previous study has found that 63% of total miles driven by trucks in 2016 could be made platoon-able, considering the speed threshold for platoon-able truck identification to be 50 mph (Lammert et al., 2018). In (Lammert et al., 2018), the authors used low-resolution data from 57,000 unique trucks for two weeks. Other literature shows a 5% to 15% reduction in fuel consumption based on the platoon configuration (Al-Qadi et al., 2021). Using Adaptive Cruise Control (ACC) or Cooperative Adaptive Cruise Control (CACC) applications to form platoons, a recent study found a 7.9% reduction in fuel consumption by 2025 and an increase in the capacity of platoon-able road segments (Noruzoliaee et al., 2021). Such advances can lead to noticeable savings of $868 million for the U.S. trucking industry, and a reduction in infrastructure improvement needs up to $4.8 billion (Noruzoliaee et al., 2021). That's why Academia and the trucking industry have been carrying out research with the aim of accelerating the broad implementation of truck platooning. Over the past few decades, there have been various partnerships between governments and private

companies that have demonstrated the use of autonomous truck platooning in practical situations. One notable example of this research is the UC Berkeley PATH program's partnership with the Volvo Group. The program demonstrated the advantages of an automated truck platoon in a real-world scenario. The study found that truck platooning could reduce fuel consumption by up to 10% for the lead vehicle and up to 15% for the following vehicles. The study also showed that platooning could improve road safety by reducing the risk of accidents caused by driver error (Tsugawa et al., 2016).

Artificial Intelligence (AI) can be used to predict each leader vehicle trajectory in autonomous truck platooning to improve platoon safety. Reliance on classical AI may not be efficient for this purpose as it will increase the computational burden for each truck in the platoon (Shladover et al., 2018). Quantum Artificial Intelligence (AI), however, can be used in this scenario to enhance learning efficiency, learning capacity, and run-time improvements (Islam et al., 2022; Dunjko & Briegel, 2018).

1.2 Hypothesis and Contribution

In this study, an algorithm was developed to predict the trajectory of the lead vehicle of an autonomous truck platoon so that the follower truck can adjust its speed and direction during a sudden change of trajectory using both Long Short Term Memory Network (LSTM) and Quantum Long Short-Term Memory Network (QLSTM). We hypothesize that the developed hybrid classical-quantum model with the mix of classical and Quantum-AI models will be more robust than the classical counterpart and less vulnerable to both (1) noise and variations in large-scale heterogeneous data and (2) inherent errors in the quantum-only approach. Though the quantum computer is in its infancy, it is expected that Quantum-AI will be more efficient in real-time autonomous truck platoon management and require less computational cost in the future. We also

hypothesize that the accuracy of Quantum-AI algorithms to predict vehicle trajectory will be comparable to classical models. In addition, this study evaluated the operational efficacy and string stability of an autonomous truck platoon with the trajectory prediction of each truck using both classical LSTM and QLSTM. We hypothesize that the platoon operating with QLSTM trajectory prediction will have better operational efficacy and string stability.

1.3 Objectives

The objectives of this study are as follows:

- Develop a trajectory prediction model of the leader vehicle of an autonomous truck platoon by using LSTM and QLSTM, and
- Evaluate the operational efficacy and string stability of the autonomous truck platoon with the trajectory prediction of each truck using both classical LSTM and QLSTM.

CHAPTER TWO

LITERATURE REVIEW

Truck platooning is the idea that two or more trucks may be linked together through automation technology and driving support systems to increase safety and efficiency. Through wireless communication, the trucks in a platoon interact with one another, enabling them to drive in close proximity to each other. The following trucks are programmed to automatically perform maneuvers such as braking, accelerating, and decelerating, in response to the actions of the lead truck. This arrangement enhances the aerodynamics of the trucks, leading to decreased fuel consumption (Patten et al., 2012). Researchers found, through testbed experiments six percent fuel savings for leader vehicles and ten percent for follower vehicles in a platoon (Alam et al., 2015; Lammert et al., 2014). Moreover, less fuel consumption led to cost savings and reduced emissions (Scora and Barth, 2006). Furthermore, a truck platoon can reduce congestion as the trucks will take less space in the platoon than driving separately (Schladoverr et al., 2015; Van Arem et al., 2006). Also, platooning can enhance traffic safety because the vehicle in platoon results in less human error and lower reaction time, thus reducing rear-end collisions. Finally, truck platooning can decrease travel time and increase roadway capacity (Lee et al., 2021).

With the advancement of automated vehicle technology, it is now possible for multiple automated trucks to travel together at a minimum safety distance or form a platoon through vehicle-to-everything (V2X) communication technology (Lee et al., 2021). With the rapid advancement of 5G and V2X communication technology, automated truck platooning is being studied by both researchers and industry (Tsugawa et al., 2016; Alam et al., 2015). As the current infrastructure cannot support a fully autonomous truck platoon, semi-automated platooning is being tested. As per the EU truck platooning roadmap, it is projected that the follower trucks within

a platoon will attain SAE Level 4 automation (automated driving without a driver) by 2025 (EAMA, 2019).

Several studies found that connected vehicle platoons can use a trajectory-tracking control model for better operational efficiency (Li et al., 2017; Chu et al., 2017). In a recent study, it was found that around 36% of truck platoons could be effectively managed by adjusting their speed, without the need to modify their routes or schedules (Ma et al., 2021). Truck platoons can make safe, efficient driving decisions with accurate road user trajectory predictions (Wei et al., 2020). Besides, with a precise prediction of the trajectory of the leader truck, the follower trucks can adjust their speed and direction during a sudden change of trajectory. Trajectory prediction can also minimize travel time, avoid congestion, and develop methods for future utilization of the road network (Yan & Shen 2019).

Truck platooning is a safety-critical application which requires trajectory prediction models for a platoon to have a high accuracy to prevent adverse consequences. Artificial intelligence (AI) has several successful applications in trajectory prediction. For example, the Artificial Neural Network (ANN) and Support Vector Machine (SVM) classifiers were used for lane change prediction, which predicted a lane change action before the actual lane change with success (Dou et al., 2016). Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) for prediction systems can process massive volumes of data. Various research fields, including trajectory prediction, are experiencing unprecedented growth due to the advent of deep learning techniques and computer capacity. Dai et al. proposed a modified LSTM model for trajectory prediction (Dai et al., 2019). Du et al. created a predictive model for the trajectory of connected vehicle platoons using a digital twin-based approach (Du et al., 2021).

Introducing a quantum algorithm into the domain of Artificial Intelligence (AI) has improved the performance of hybrid machine-learning models (Dunjko & Wittek, 2020). In Artificial Intelligence (AI), quantum computing generates multiple states with measurable performance essential for transfer learning (Bassman et al., 2021). Similarly, quantum annealing, and quantum random walk offer optimization from the previously suggested multiple guesses. Usually, it generates NP-hard solutions (Crosson & Harrow, 2016), and Quantum AI techniques have been developed to cope with these problems (Sgarbas, 2007). The capacity of Quantum Neural Networks (QNNs) to extract solutions from intricate probability distributions is what makes them useful in machine learning models. This is achieved by encoding information into a quantum state through a quantum feature map, which is a type of variational quantum algorithm (Abbas et al., 2021). With the extension of AI and machine learning, Quantum deep learning has also gained renown in solving intractable problems on regular classical computers (Wiebe et al., 2014). Patel et al. (2019) applied a Quantum Neural Network (Q-NN) for signature verification, and the accuracy was 95% compared to the classical neural network (CNN), which achieved 89% accuracy (Patel et al., 2019). Furthermore, Patel & Tiwari (2014) utilized Quantum Binary NN (Q-BNN) model for breast cancer classification and compared it against Gaussian Processes, NNs, Multilayer Perceptron (MLP), Support Vector Machine (SVM). Q-BNN achieved above 95% accuracy, whereas other methods were less than 80% accurate (Patel & Tiwari, 2014). Chen et al. (2020) applied a Quantum Convolutional Neural network (Q-CNN) for image classification and reported higher accuracy (94%) than classical CNN (90%) (Chen et al., 2020). Wang et al. (2022) showed that Quantum Stochastic Networks (Q-SNN) could achieve better performance against classical networks classifying sentences (Wang et al. 2022). Q-SNN converged faster and with higher accuracy compared to classical SNN. Quantum computing applications in AI have been

beneficial in many fields, such as in operational optimization (Azad et al., 2022; Zhang et al., 2020), transportation systems cyber-security (Khan et al., 2021), and human traffic intention estimation and trajectory prediction (Busemeyer & Bruza, 2012; Song et al., 2022). In autonomous vehicles and quantum computers developments, previous assumptions regarding the interaction between autonomous vehicles and pedestrians being classical in the sense that behavior is rational are no longer sacrosanct. It is now assumed to follow the quantum decision theory, making human behavior irrational, and violating classical cognitive and decision theory (Song et al., 2022). Academics have concluded that the interplay between interference and entanglement in quantum mechanics and human cognition shares several common traits (Busemeyer & Bruza, 2012). This observation has resulted in a prevailing trend.

Car-following models regulate a driver's actions in relation to the vehicle directly in front of them in the same lane (Brackstone and McDonald, 1999). There are five categories in which car-following models can be classified: the Gazis-Herman-Rothery (GHR) model, the Collision Avoidance (CA) model, the Linear Model, the Fuzzy-logic-based model, and the Optimal Velocity (OV) model, including its variations (Panwai and Dia, 2005; Brackstone and McDonald, 1999). One of the first and most advanced models for cars that followed is the Gazis-Herman-Rothery (GHR) model. The model has the drawback of having characteristics that change depending on the driving environment. Similar to the GHR model, the linear model has been extensively investigated; however, although having a very straightforward and linear shape, it is less widely used due to the challenges associated with parameter calibration. Given the characteristics of car-following behaviors, fuzzy logic seems like a realistic attempt to incorporate into the car-following theory. However, the usefulness of such efforts is constrained by the challenge of calibrating the membership function, which is the fundamental idea of fuzzy logic. However, the most popular

car-following model for simulation is arguably Gipps' adaptation of the Collision Avoidance (CA) model. A very recent car-following model, the Optimal Velocity (OV) concept, was initially put forth in 1990. The model is distinctive in how it depicts stop-and-go and backed-up traffic. Later, two OV model variations, named Generalized Force (GF) model and Full Velocity Difference (FVD) model, were developed to address OV model problems with data agreement and startup process.

CHAPTER THREE

METHOD

The following subsections provide a description of the various steps of the method, as presented in Figure 3.1.

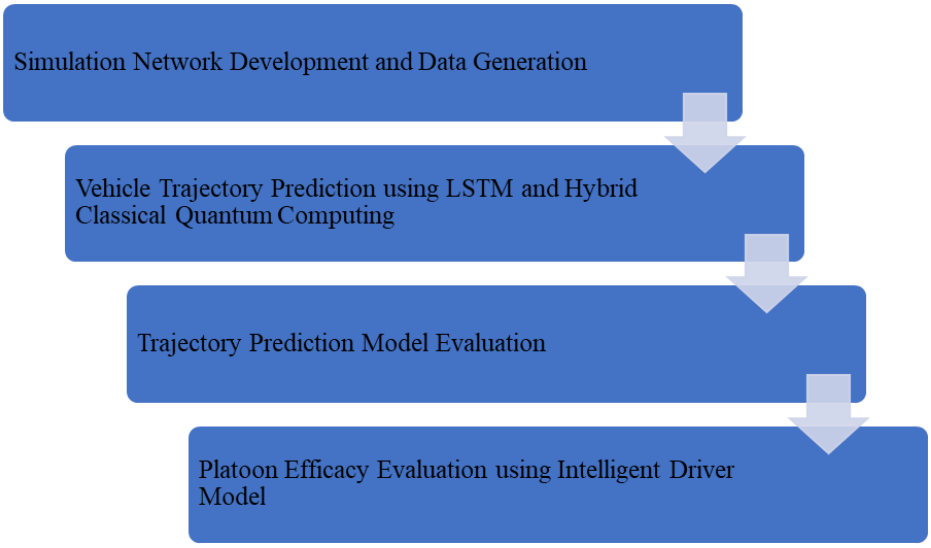

Figure 3.1 Research Method

3.1 Simulation Network Development and Data Generation

For a simulation duration of 900 seconds, we utilized MATLAB to simulate a platoon of five automated trucks employing Cooperative Adaptive Cruise Control (CACC). This platoon consisted of one leader and four follower trucks, and it was derived from (Salek, 2022). The leader truck starts at a distance of 163 meters, and the other trucks are placed at distances of 127, 91, 56, and 20 meters from the starting point, respectively. All five trucks are moving at an initial speed of 31.44 m/s, or 70.3 mph. Input parameters for the simulation include the number of trucks in the platoon, the total simulation time, the initial position and speed of the follower trucks, the location

and speed profile of the lead truck, and the constant required time headway. A simulation step size of 0.1 sec and a constant desired time headway of 0.5 sec were used. Figure 3.2 and Table 3.1 show the speed profile of the leader truck from 0 sec to 900 sec.

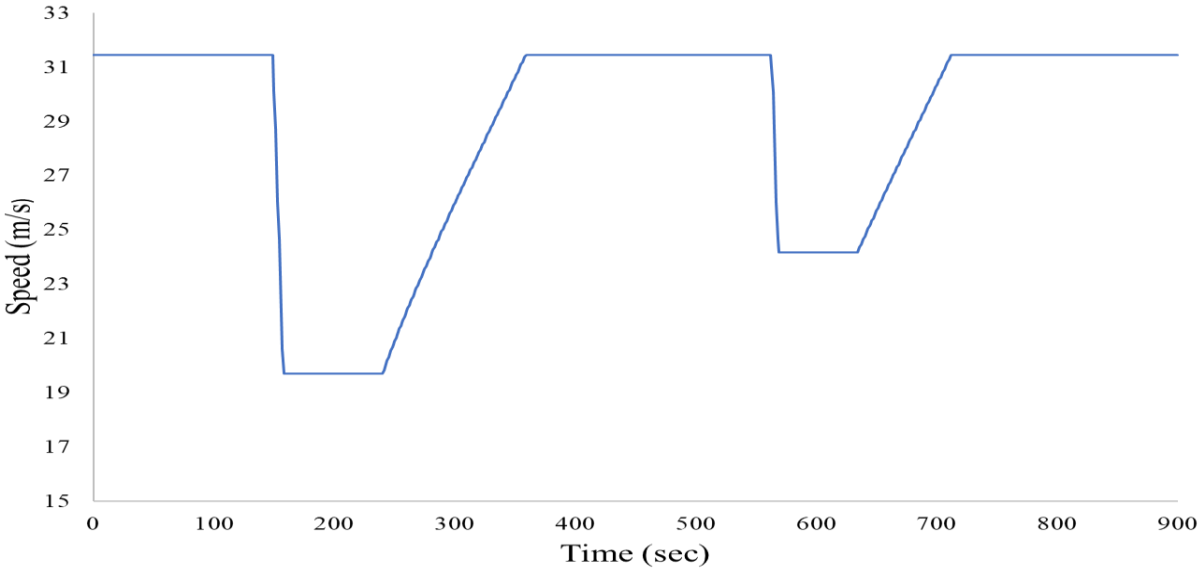

Figure 3.2: Speed Profile of the Leader Truck

TABLE 3.1 Speed Profile of the Leader Truck

Time Interval	**Speed Profile of the Leader Truck**
0 s to 149 s	Uniform speed of 31.44 m/s (70.3 mph)
149 s to 158 s	Speed changes from 31.44 m/s (70.3 mph) to 19.69 m/s (44.04 mph)
158 s to 240 s	Uniform speed of 19.69 m/s (44.04 mph)
240 s to 359 s	Speed changes from 19.69 m/s (44.04 mph) to 31.44 m/s (70.3 mph)
359 s to 562 s	Uniform speed of 31.44 m/s (70.3 mph)
562 s to 569 s	Speed changes from 31.44 m/s (74.3 mph) to 24.15 m/s (54.02 mph)
569 s to 634 s	Uniform speed of 24.15m/sec (54.02 mph)
634 s to 712 s	Speed changes from 24.15 m/s (54.02 mph) to 31.44 m/s (70.3 mph)

712 s to 900 s	Uniform speed of 31.44 m/s (70.3 mph)

The I-26 freeway in South Carolina's Berkeley, Orangeburg, and Dorchester County was the site of a calibrated traffic simulation network that Rahman et al. used to determine the speed profile of the leader truck (Rahman et al., 2015). The PTV VISSIM traffic simulation software was used to design the I-26 roadway network. It was calibrated using field data to simulate traffic volumes and trip durations within 10% of the actual data on those variables. On the VISSIM I-26 network, there are two areas with reduced speed limits of 55 and 45-mph. The 75-mph limit applies to the remaining sections of the simulated I-26 network. The following assumptions are made for the simulated truck platoon:

- The truck platoon operates in the same lane,
- All the trucks of the platoon have the exact dimensions and vehicle dynamics,
- There are no vertical or horizontal curves in the section,
- After the platoon formation, no trucks attempt to join the platoon or exit the platoon,
- There is no noticeable delay in the following trucks' real-time receipt of their immediate neighboring trucks' location and speed information.

To simulate the platoon consisting of one leader truck and four following trucks, we solved a group of first-order differential equations in MATLAB. We followed the methodology presented by Rahman et al. to create a system of first-order differential equations and utilized the "ode45" MATLAB solver (Rahman et al. 2017). Finally, the trajectory dataset was generated for all five trucks from timestamp 0 sec to 900 sec. As shown in Table 3.2, the dataset contains the following

fields: (i) timestamp, (ii) X_pos (absolute X coordinate of the vehicle), (iii) Distance (distance covered by the vehicle in m), and (iv) Speed (speed of the vehicle in m/s).

TABLE 3.2: Vehicle Trajectory Dataset

Time	**X_Pos**	**Distance**	**Speed**
0	162.8806	0	31.44032
0.1	166.0247	3.144032	31.44032
0.2	169.1687	3.144032	31.44032
0.3	172.3127	3.144032	31.44032
0.4	175.4568	3.144032	31.44032

At first, we divide the trajectory dataset into two separate datasets. One dataset (from the time stamp 0 sec to 450 sec) was used for model development, and another dataset (from the time stamp 451 sec to 900 sec) was used to evaluate the model. Then, the dataset used for model development was split into the train dataset (70%) and test dataset (30%)

3.2 Vehicle Trajectory Prediction using LSTM and Hybrid Classical Quantum Computing

LSTM (Long Short-Term Memory) is a recurrent Neural Network (RNN) applicable to a broad range of problems aiming to analyse or classify sequential data. LSTM can be used to predict the speed of the leader vehicle of a platoon based on the historical data sequences with great success. LSTM uses a certain number of past observations to predict the future. Sequence Length is the deciding factor in choosing the number of observations the LSTM considers in advance. If the sequence length is n, then the LSTM considers the last n observations to predict the $(n+1)^{th}$ observation. In this study, a sequence length of 3 was used. We used the train dataset to

train the LSTM model, and the test dataset to test the model. A learning rate of 0.0001 that gave accurate results was decided after some experimentation. The number of epochs used was 20.

The LSTM's efficiency and trainability can be improved by replacing some of the layers in the LSTM with variational quantum layers, a quantum-classical hybrid model of LSTM. Quantum LSTM (QLSTM) can learn significantly more information after the first training epoch than its classical counterpart and better learning capability of the local features while having fewer parameters than classical LSTM (Chen et al., 2020). We used the same datasets for classical LSTM model development and evaluations for QLSTM model development and evaluation. This study used Pennylane-enabled variational quantum layers to replace the LSTM layers. The following variational quantum circuits shown in Figure 3.3 serve as the foundation for the variational quantum layers:

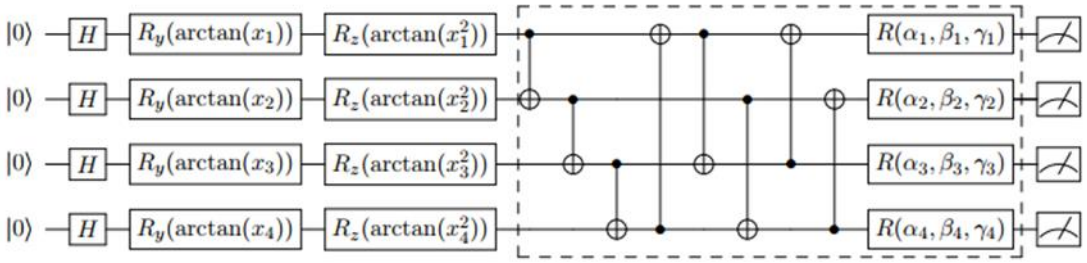

Figure 3.3: VQC Architecture for QLSTM (adapted from Chen et al., 2020)

These variational quantum circuits were run on the built-in Pennylane simulator. This study used four qubits, one variation layer, and a learning rate of 0.05. The parameters were chosen through experimentation on fewer epochs to see which produces the best outcomes.

3.3 Trajectory Prediction Model Evaluation

Three performance metrics were used to evaluate the models: Mean Average Error (MAE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE). The MAE calculates the average of the absolute differences between predicted and actual values. The RMSE calculates the square root of the average squared differences between predicted and actual values. The MAPE represents the average of absolute percentage errors. The performance matrices can be measured from the following Equations:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \quad (1)$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y_i - x_i)^2}{n}} \quad (2)$$

$$MAPE = \left(\frac{\sum_{i=1}^{n} \left|\frac{y_i - x_i}{y_i}\right|}{n} \times 100\right) \quad (3)$$

Where $y_i = predicted\ value\ of\ the\ i^{th}\ sample, x_i = observed\ values\ of\ the\ i^{th} sample$ $and\ n = number\ of\ samples$

3.4 Platoon Stability Evaluation using Intelligent Driver Model

Finally, this study evaluated the automated truck platoon's stability with trajectory prediction from classical LSTM and QLSTM using Intelligent Driver Model (IDM) by comparing the following:

- Speed Profiles
- Inter-truck gap profiles
- Jerk profiles

A simplified version of IDM for acceleration (Treiber et al., 2000) of the control vehicle can be expressed as,

$$a_c = a\left[1 - \left(\frac{v_c}{v_{des}}\right)^{\delta} - \left(\frac{d^*(v_c,\ \Delta v_l)}{d_l}\right)^2\right] \quad (4)$$

where

a is the normal acceleration,

Δv_l is the gap between the control vehicle and the leading vehicle,

$d^*(v_c,\ \Delta v_l)$ is the desired gap between the control vehicle and its leading vehicle, and

δ is an exponent for the vehicle's acceleration.

The acceleration of the control vehicle in the IDM model has two parts: $1 - \left(\frac{v_c}{v_{des}}\right)^{\delta}$ accounts for the desired acceleration of the control vehicle and $\left(\frac{d^*(v_c,\ \Delta v_l)}{d_{des}}\right)^2$ accounts for the braking deceleration of the control vehicle when its immediate leading vehicle is decelerating. The desired gap $d^*(v_c,\ \Delta v_l)$ is defined as follows,

$$d^*(v_c,\ \Delta v_l) = d_{min} + \max\left[0, \left(d_{des} + \frac{v_c \Delta v_l}{2\sqrt{ab}}\right)\right] \quad (5)$$

where

d_{min} is the minimum gap to be maintained between two vehicles, and

b is the normal comfortable braking deceleration.

For the N-vehicle simulation scenario, the acceleration of the N-th vehicle can be written as,

$$\ddot{x_n} = a\left[1 - \left(\frac{\dot{x_n}}{v_{des}}\right)^{\delta} - \left(\frac{d^*(\dot{x_n},\ \Delta\dot{x_n})}{x_{n-1} - x_n - l}\right)^2\right] \quad (6)$$

where

$$\Delta\dot{x_n} = \dot{x_{n-1}} - \dot{x_n} \quad (7)$$

$$d^*(\dot{x_n},\ \Delta\dot{x_n}) = d_{min} + \max\left[0, \left(d_{des} + \frac{\dot{x_n}\Delta\dot{x_n}}{2\sqrt{ab}}\right)\right] \quad (8)$$

The string stability of the platoon was also evaluated by giving the predicted trajectory of the platoon obtained from the LSTM and QLSTM as input in the Intelligent Driver Model. The sum of squared speed error (SSSE) was used as an evaluation matrix (Salek et al., 2022). SSSE at a given timestamp can be calculated from the following equation:

$$SSSE\ (t_i) = (v_L(t_i) - v_1(t_i))^2 - \sum_{j=2}^{N}(v_{j-1}(t_i) - v_j(t_i))^2 \quad (9)$$

where $T_{h,j}(t_i)$ is the actual time headway of the j^{th} follower truck at $t_i$with its immediate leading truck, $v_L(t_i)$ is the leader truck's speed at t_i, and $v_j(t_i)$ is the j^{th} follower truck's speed at t_i.

For an autonomous truck platoon to be considered string stable, the non-zero speed errors of any truck should not be amplified in the preceding or following trucks (Bose and Ioannou, 2001; Pueboobpaphan and Van Arem, 2010). A lower value of SSSE indicates better string stability. We compared the SSSE values to determine which model demonstrated higher string stability for the autonomous truck platoon.

CHAPTER FOUR

ANALYSIS

The comparison of training loss and testing loss for the LSTM and the QLSTM is shown in Figure 4.1 and Figure 4.2, respectively. The QLSTM learns significantly more information for the training loss in the early epochs than the LSTM, and its results converge much faster than its classical counterpart. Similarly, we can see that the QLSTM learns substantially more information than the LSTM in the first few epochs for the testing loss and converges to a lower value more quickly. It proves that the QLSTM has better trainability than the LSTM.

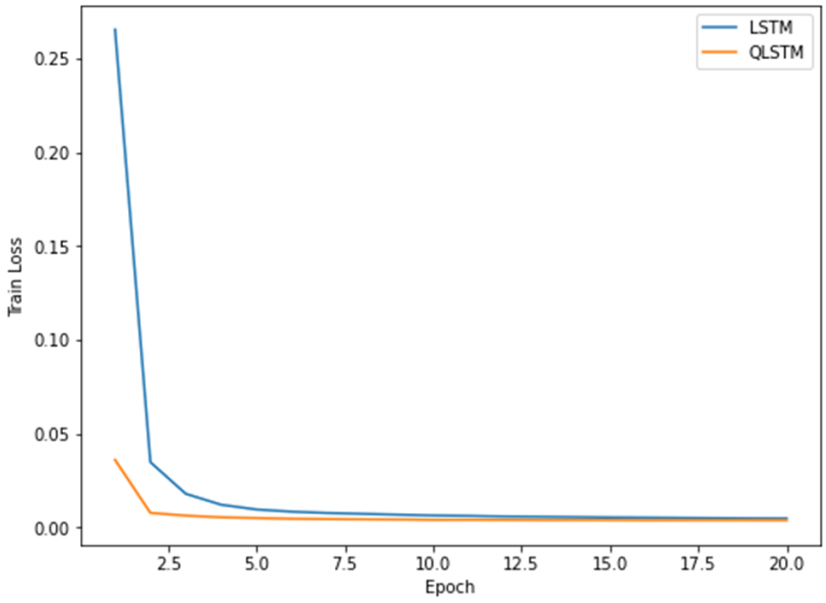

Figure 4.1: Comparison of training loss

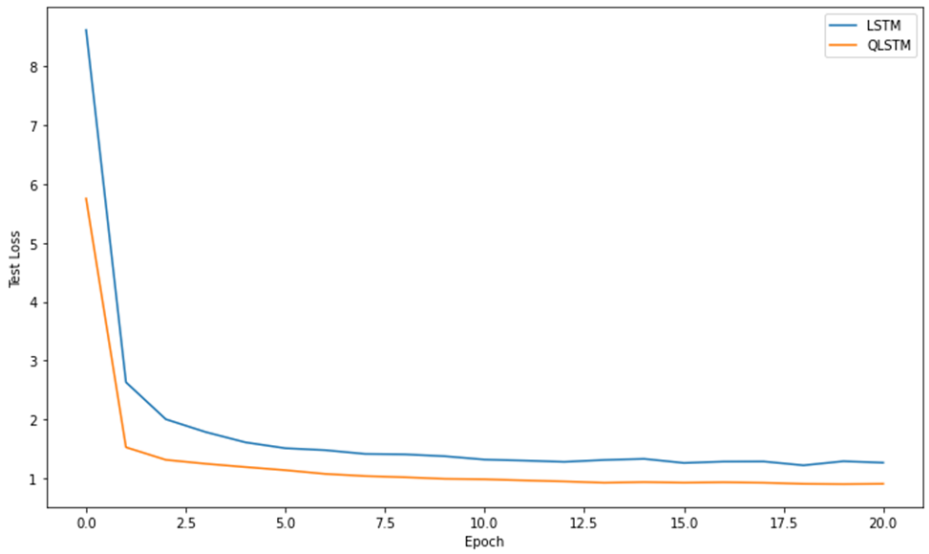

Figure 4.2: Comparison of testing loss

Figure 4.3 shows a comparison of the Mean Absolute Error for the trajectory prediction with LSTM and QLSTM for each truck in the autonomous truck platoon. Predicted trajectories that use QLSTM have 4% to 8% less mean absolute error than the predicted trajectories that use LSTM.

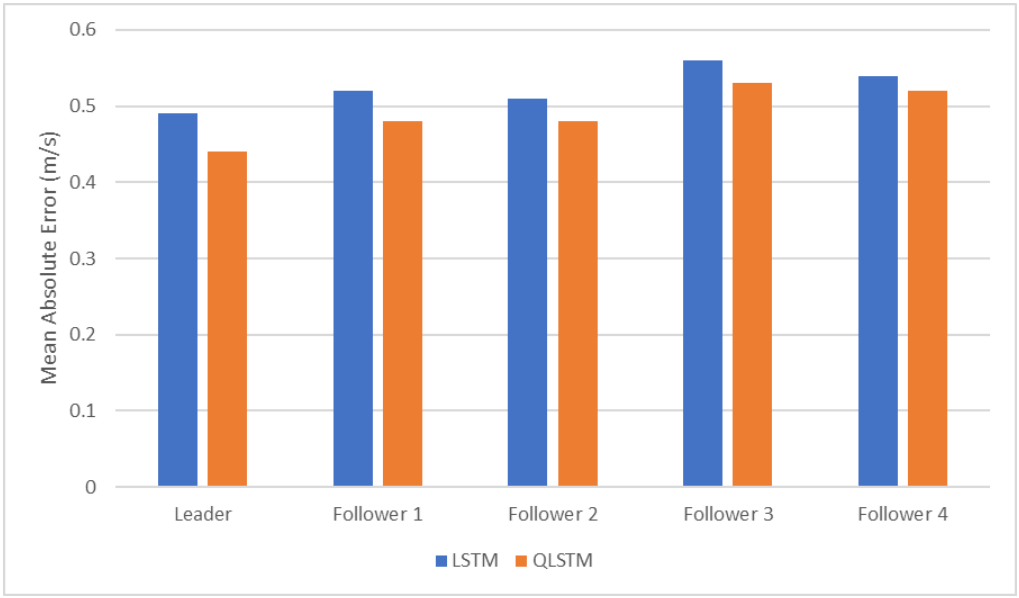

Figure 4.3: Comparison of Mean Absolute Error

Figure 4.4 shows a comparison of the root mean squared error for the trajectory prediction with LSTM and QLSTM for each truck in the autonomous truck platoon. Predicted trajectories that use QLSTM have 3% to 6% less root mean squared error Predicted trajectories that use LSTM.

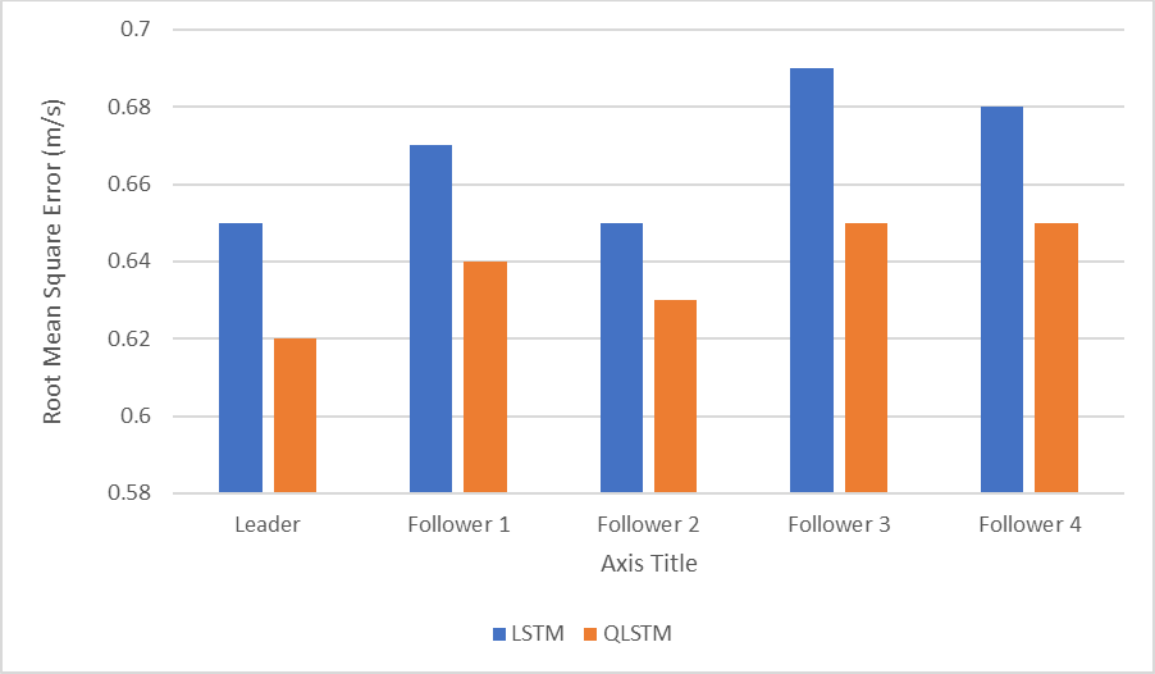

Figure 4.4: Comparison of Root Mean Squared Error

Figure 4.5 shows a comparison of the Mean Absolute Percentage Error for the trajectory prediction with LSTM and QLSTM for each truck in the autonomous truck platoon Predicted trajectories that use QLSTM have almost the same mean absolute percentage error as the Predicted trajectories that use LSTM.

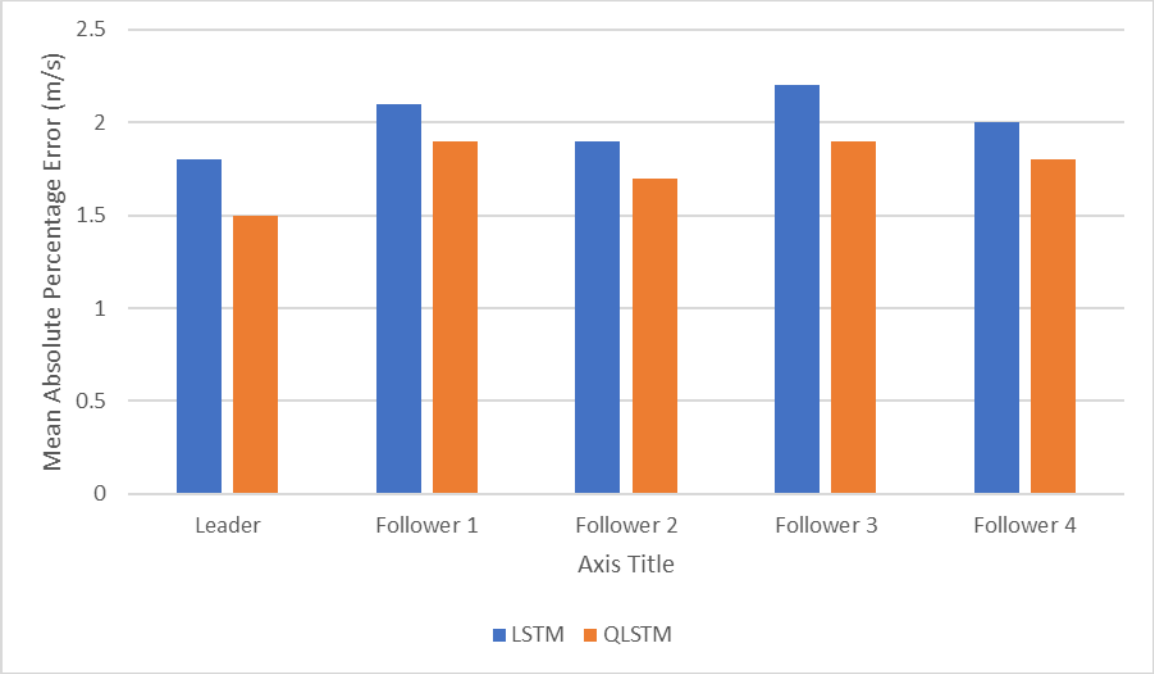

Figure 4.5: Comparison of Mean Absolute Percentage Error

Figure 4.6 shows the speed profile of every truck in the platoon for the original trajectory and figures 4.7-4.8 6 present the speed profile of every truck in the platoon for predicted trajectory by LSTM and QLSTM, respectively. The situations that require critical evaluation involve when the leader truck enters areas with lower speed limits and applies brake to keep its speed within the reduced speed limit. In all three scenarios, it was noted that every follower truck could closely match the leader truck's speed profile throughout the whole simulation. From the figures we see that, the speed profiles of autonomous trucks using LSTM and QLSTM prediction for trajectory is almost same of the speed profiles with original trajectory. From this finding it can be inferred that the predicted trajectory using both LSTM and QLSTM can be used for truck platooning operations when original trajectory is not available.

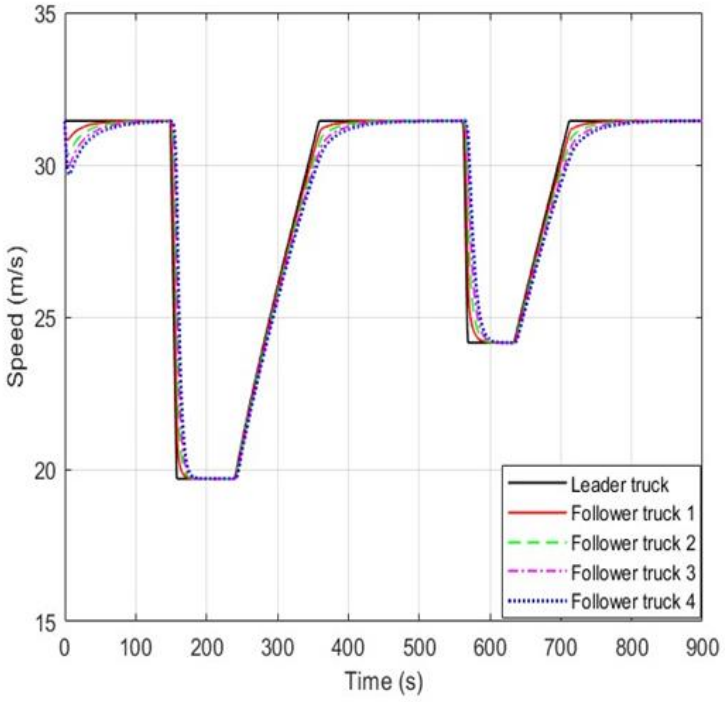

Figure 4.6: Speed profiles of autonomous trucks with original trajectory

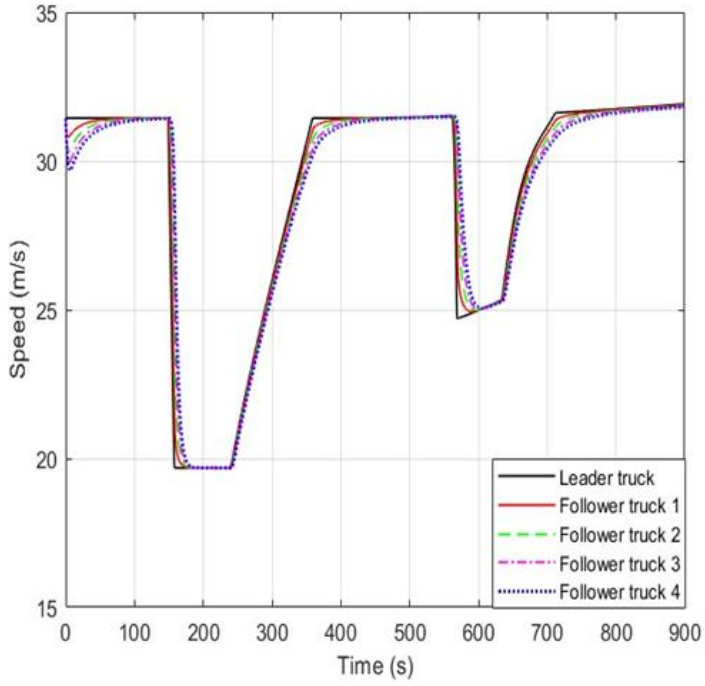

Figure 4.7: Speed profiles of autonomous trucks using LSTM trajectory prediction

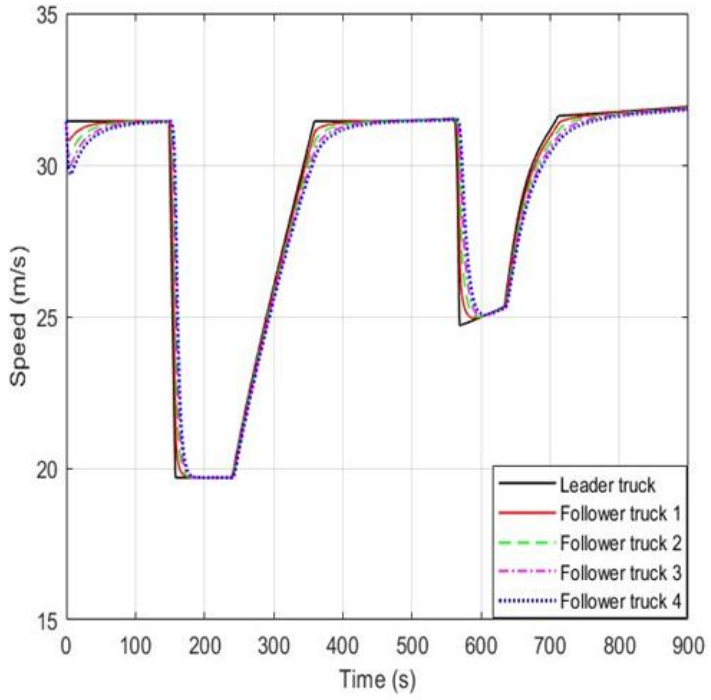

Figure 4.8: Speed profiles of autonomous trucks using QLSTM trajectory prediction

Figures 4.9 presents the inter-truck gap profiles between every two trucks in the platoon for the original trajectory and figures 4.10-4.11 show the inter-truck gap profiles between every two trucks in the platoon for predicted trajectory by LSTM and QLSTM, respectively. From the inter-truck gap profiles, it can be deduced that there is no risk of a collision between the trucks in the platoon because none of the inter-truck gaps exhibit zero or a negative value. Also, the inter-truck gap profiles demonstrate that each following vehicle can consistently maintain a safe distance of at least 10 meters from the truck directly in front of it. Overall, the platoon's followers maintain uniform spacing for all the three scenarios. From the figures we see that, the inter-truck gap profiles of autonomous trucks using LSTM and QLSTM prediction for trajectory is almost same of the inter-truck gap profiles with original trajectory. It's because of the higher accuracy in prediction of trajectory of the autonomous trucks.

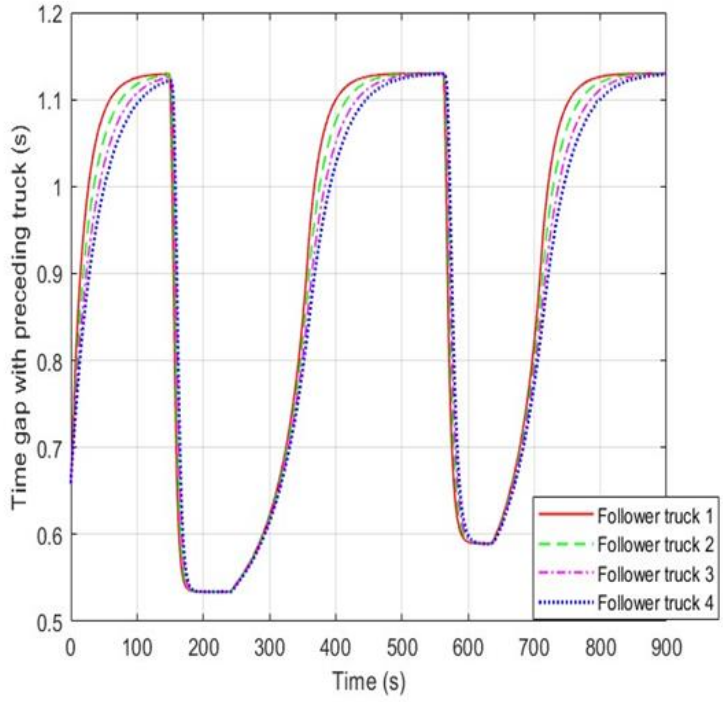

Figure 4.9: Inter-truck gap profiles of automated trucks with original trajectory

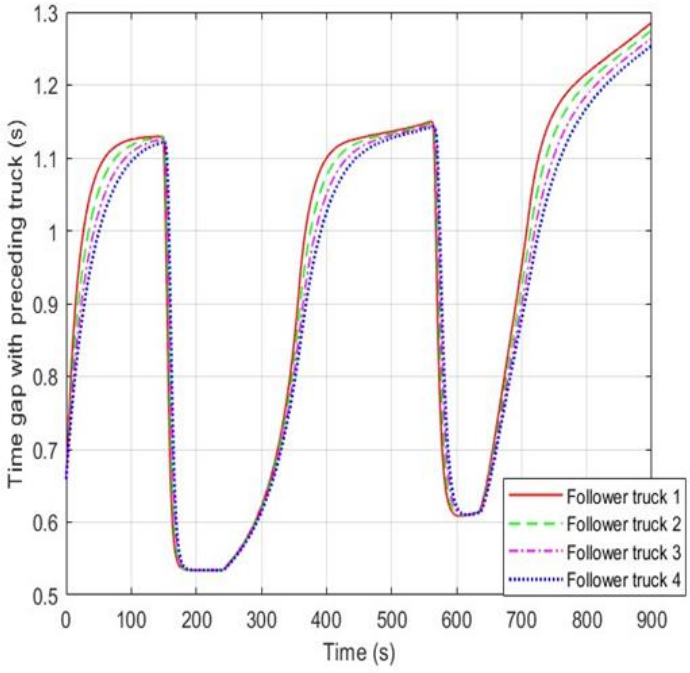

Figure 4.10: Inter-truck gap profiles using LSTM trajectory prediction

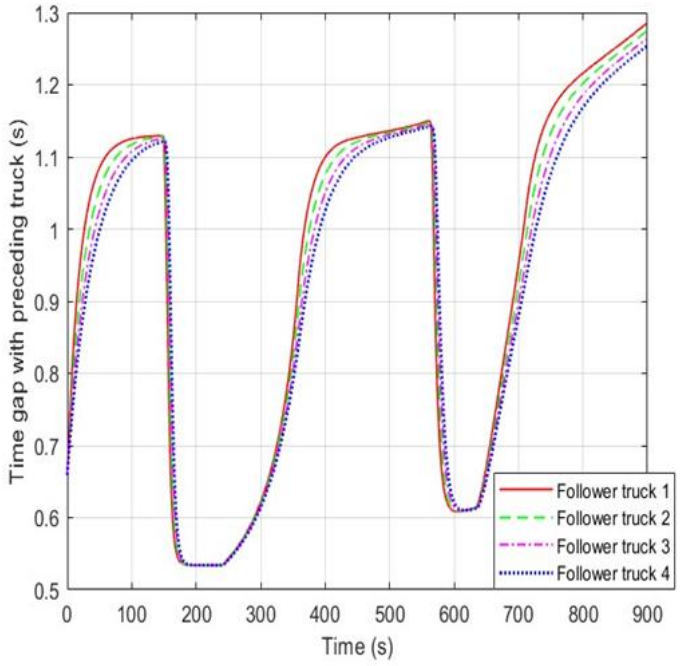

Figure 4.11: Inter-truck gap profiles using QLSTM trajectory prediction

Figure 4.12 shows jerks for every truck in the platoon for the original trajectory and figures 4.13-4.14 present jerks for every truck in the platoon for the predicted trajectory by LSTM and QLSTM, respectively. From the figures we can see that jerk is higher for the two short intervals that follow the two non-linear deceleration stages (during which the follower trucks attempt to resume uniform speed) in all three scenarios. Except for the two brief intervals that follow the two non-linear deceleration stages (during which the follower trucks attempt to resume uniform speed), the jerk is almost zero for all three scenarios. From the figures it is evident that, the jerk profiles of autonomous trucks using LSTM and QLSTM prediction for trajectory is almost same of the jerk profiles with original trajectory which denotes the higher accuracy in prediction of trajectory using both LSTM and QLSTM.

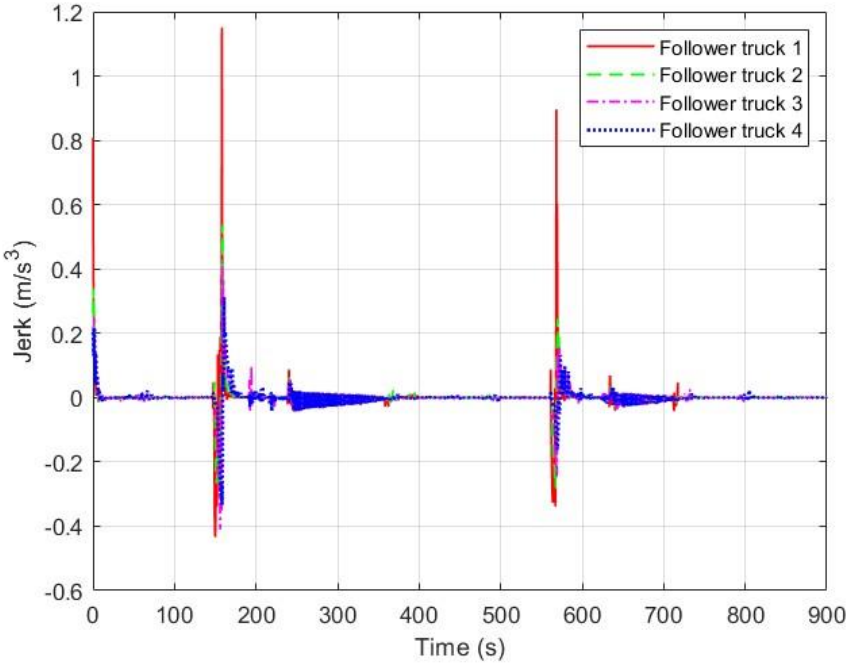

Figure 4.12: Jerk profiles of autonomous trucks with original trajectory

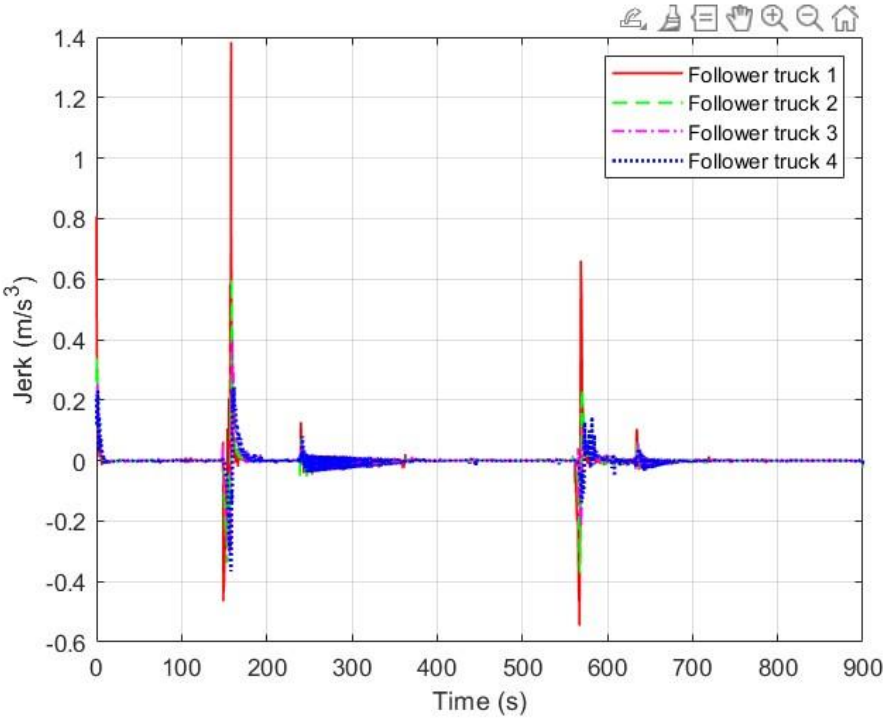

Figure 4.13: Jerk profiles of autonomous trucks using LSTM trajectory prediction

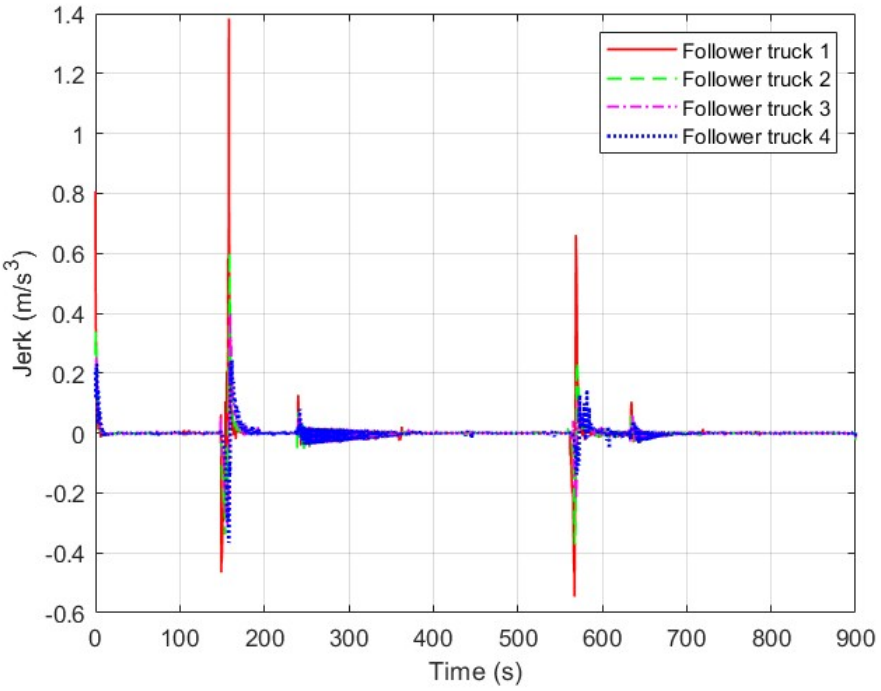

Figure 4.14: Jerk profiles of autonomous trucks using QLSTM trajectory prediction

Figures 4.15 and 4.16 present the SSSE profiles for LSTM and QLSTM trajectory prediction, respectively. We used the SSSE profiles to evaluate the string stability of the autonomous truck platoon. Here, the platoon using QLSTM trajectory prediction had lower SSSE than the platoon using LSTM. The figures show that both scenarios lead to SSSE values near 0, except for the two brief periods after the two non-linear deceleration phases, where the following trucks strive to return to a uniform speed. Furthermore, even after these two brief intervals, the SSSE of the QLSTM remains lower than that of the LSTM. Thus, the QLSTM offers better string stability than the LSTM in all traffic conditions.

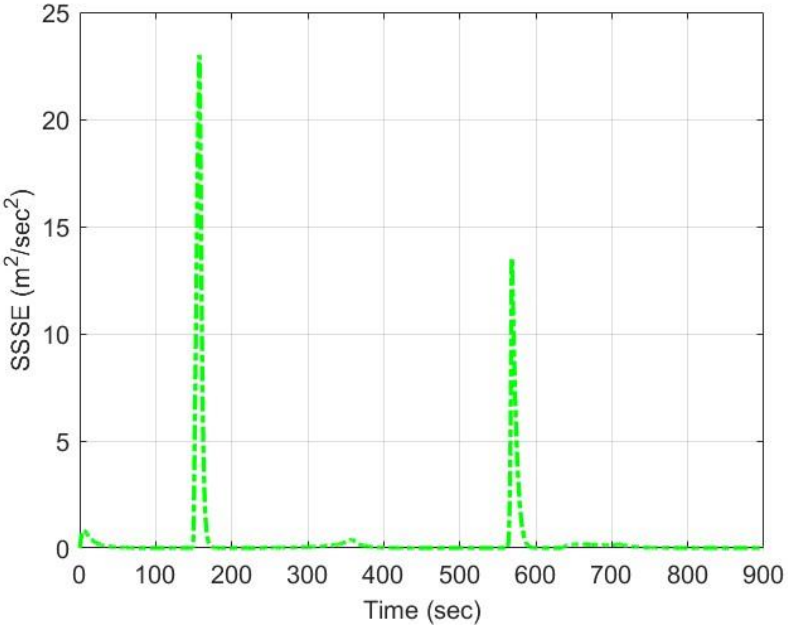

Figure 4.15: SSSE profiles of autonomous trucks using LSTM trajectory prediction

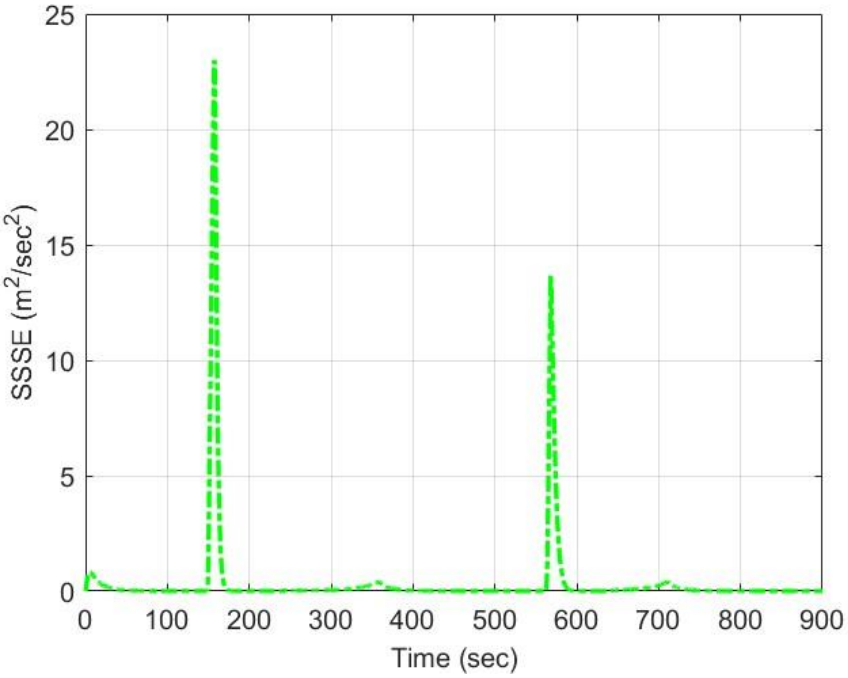

Figure 4.16: SSSE profiles of autonomous trucks using QLSTM trajectory prediction

Although the results from LSTM and QLSTM are almost the same, the QLSTM used fewer parameters (181) than classical LSTM (1425). So, it can be inferred that the QLSTM will require less data for predicting the trajectory of the trucks in an autonomous truck platoon.

CHAPTER FIVE

CONCLUSIONS

I(n this study, we used MATLAB to simulate a platoon of five autonomous trucks (with one leader truck and four following trucks) for a duration of 900 seconds. A set of first-order differential equations was solved in MATLAB to mimic the platoon of five trucks. Finally, the trajectory dataset was generated for all five trucks from timestamp 0 sec to 900 sec. Then, the study developed and evaluated an LSTM model as well as a QLSTM for predicting the trajectory of the leader vehicle of an automated truck platoon. This study also evaluated an autonomous truck platoon's operational efficacy and string stability of the autonomous truck platoon with the prediction of trajectory from both classical LSTM and QLSTM using the IDM.

5.1 Summary of Findings

The analysis found that both the LSTM and QLSTM gave comparable results. It can be inferred that Quantum-AI will be more efficient in real-time management and require less computational burden for an Automated Truck Platoon. The QLSTM learned significantly more about the training loss in the early epochs than the LSTM, and its results converge much more quickly than its classical counterpart. Moreover, QLSTM also used fewer parameters compared to classical LSTM. Besides, the QLSTM learns substantially more information than the LSTM in the first few epochs for the testing loss and converges to a lower value more quickly. In addition, this study also evaluated the operational efficacy and string stability of the autonomous truck platoon with trajectory prediction from both classical LSTM and QLSTM using the IDM. The platoon operating with LSTM and QLSTM trajectory prediction showed comparable operational

efficiency. Moreover, the platoon operating with QLSTM trajectory prediction provided better string stability.

This study showed that, QLSTM can be used very effectively to predict the speed trajectory of the leader truck in an automated truck platoon, producing results that are on par with those of its classical counterpart while requiring significantly fewer training parameters and yielding more data per epoch. It is expected that with the development of quantum computers, hybrid quantum-classical artificial intelligence would become more efficient in real-time management and require less computational burden for an autonomous truck platoon.

5.2 Limitations and Future Research Direction

The following subsections present the study's limitations for this thesis and future research direction.

5.2.1 Limitations

In a real quantum computer, in some scenarios, it may be required to reduce the number of gates in a quantum circuit is required to improve the overall efficiency and speed of the computation, but it may also come at the cost of some accuracy. In this study, we used pennylane simulator which acts as an ideal quantum computer free from any errors. Another limitation of this study is that the automated platoon formation didn't consider trucks entering and exiting the platoon. We also did not consider the lateral movement of the trucks in the platoon. Our future study will focus on predicting the trajectory of the leader vehicle of an automated truck platoon with both longitudinal and lateral movement as well as trucks entering and exiting the platoon. Currently, the model doesn't consider the heterogeneity of vehicles and existing communication

delay present in a real-world setting. Future studies will also evaluate the efficacy of the trajectory prediction algorithm in the real-world environment using real automated trucks.

5.2.2 Future Research Directions

The author recommends the following to advance the study presented in this thesis:

- An application development platform can be developed for implementing the approach developed in the study to help researchers and developers implement the autonomous truck platooning strategy.
- Future research can be extended to utilize the evolving power of quantum computers to improve the prediction efficacy of QLSTM.
- The current research can be extended to use real-world autonomous truck platoon data to evaluate the comparative efficacy of LSTM and QLSTM in the performance of autonomous truck platoon.
- A future study should evaluate autonomous truck platoon performance for different types of autonomous vehicle controllers besides IDM.
- A field evaluation with an actual autonomous truck platoon is recommended using the approach presented in this thesis for different traffic conditions.

.

APPENDICES

APPENDIX A

MATLAB CODE FOR TRAJECTORY GENERATION IN IDM

```
# IDM Model

function [dy] = idm(t,y,N,Tfinal,Xfinal,Vfinal, Th_const)

dy = zeros(2*N,1);

%--------parameter

x0_dot =33.2; % unit is m/sec

delta =4;

a=2;

b=2.94;

kk = 1/(2*sqrt(a*b));

lc=15;

Th=Th_const; % Time headway

s0=0; % minimum gap =2.0 meters

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x0 = interp1(Tfinal,Xfinal,t);

% v  = diff(x0);

v  = interp1(Tfinal,Vfinal,t);


Delta_v(1) = y(N+1)-v;

s(1) = x0-y(1) - lc;

s_star(1) = s0 + max(0,y(N+1)*Th + y(N+1)*Delta_v(1)*kk);
```

```
for i=2:N
    Delta_v(i) = y(N+i)-y(N+i-1);
    s(i)       = y(i-1)-y(i) - lc;
    s_star(i)  = s0 + max(0,y(N+i)*Th + y(N+i)*Delta_v(i)*kk);
end
%-main system-- %%%%
for i=1:N
    dy(i) = y(i+N);
end
for i=1:N
    dy(i+N) = a*(1 - (y(i+N)/x0_dot).^delta - (s_star(i)/s(i)).^2);
end


end
```

```
# Codes for Leader Vehicle Trajectory Generation


close all; clear all; clc;


%%%%%%%%%% INPUT PARAMETERS  %%%%%%%%%%
l = 15;                          % Length of trucks
s0 = 5;                          % Minimum safety gap
Th_const = 0.4;                  % Constant time gap
```

```
N_follower = 4;                    % Number of follower trucks
%%%%%%%%%% Initialize the leader truck's trajectory %%%%%%%%%%
Xfinal = zeros(0,0);               % Position
Tfinal = zeros(0,0);               % Timestamps
Vfinal = load('Vfinal.txt');       % Speed

F_time = size(Vfinal,1) - 1;       % Total time
xb = [0:1:F_time]';
xa = [0:0.1:F_time]';
Vfinal = interp1(xb, Vfinal, xa);  % Interpolate speed

%%%%%%%%%% Determine the leader truck's trajectory %%%%%%%%%%
k = 0 ;
N = N_follower + 1;
init_gap = 31.4403232*Th_const + l + s0;

for i=0:0.1:F_time
   if i == 0
      s1 = init_gap*N;
      X = s1;
      Xprevious = s1;
   else
      s2 = ((Vfinal(k+1,1) + Vfinal(k,1))/2)*0.1;
```

```
        X = Xprevious + s2;
        Xprevious = X;
    end

    Xfinal=[Xfinal;X];
    Tfinal=[Tfinal;i];

    k = k +1;
end

%%%%%%%%%%%% Save trajectory %%%%%%%%%%%%%%
writematrix(Tfinal, 'T.txt');
writematrix(Xfinal, 'X_L.txt');
writematrix(Vfinal, 'V_L.txt');

# Codes for Follower Vehicle Trajectory Generation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%

close all; clear all; clc;

%%%%%%%%%%% INPUT PARAMETERS  %%%%%%%%%%%
l = 15;                   % Length of trucks
```

```
s0 = 5;                    % Minimum safety gap
Th_const = 0.5;            % Constant time gap
N_follower = 1;            % Number of follower trucks to be geenrated

%%%%%%%%%% Load the preceeding truck's trajectory %%%%%%%%%%%%
Xfinal = load('X_L.txt');     % Position
Tfinal = load('T.txt');     % Timestamps
Vfinal = load('V_L.txt');     % Speed

%%%%%%%%%% DERIVED PARAMETERS  %%%%%%%%%%
N = N_follower + 1;
F_time = Tfinal(end);
init_gap = 31.4403232*Th_const + l + s0;

%%%%%%%%%%%% Initial Position and Speed %%%%%%%%
initial_values = zeros(2*N,1);

for i = 1:N
   initial_values(i) = Xfinal(1,1) - init_gap*i;
   initial_values(i+N) = Vfinal(1,1);
end

%%%%%%%%%%%%%%%%%%%% SOLUTION OF SYSTEM OF EQUATIONS  %%%%%
```

```
tspan = 0:0.001:F_time;
[T,Yidm] = ode45( @(t,y) ...
    idm(t, y, N, Tfinal, Xfinal, Vfinal, Th_const), tspan, initial_values);

%%%%%%%%%% Determine the follower truck's trajectory %%%%%%%%%%%%

xa = [0:0.1:F_time]';

X_f = interp1(tspan,Yidm(:,1),xa);
V_f = interp1(tspan,Yidm(:,N+1),xa);

%%%%%%%%%% Plot IDM ---Speed %%%%%%%%%%%%%%

figure(1);
hold on;
builtin('plot',Tfinal,Vfinal,'k-','LineWidth', 1)
plot(Tfinal,V_f,'r-','Linewidth',1)

xlabel('Time (sec)')
ylabel('Speed (m/sec)')
xlim([0 900])
ylim([15 40])

legend([builtin('plot',Tfinal,Vfinal,'k-','LineWidth', 1) ...
    plot(Tfinal,V_f,'r-','Linewidth',1)], ...
```

```
    'Preceeding truck','Follower truck');
grid on
box on

%%%%%%%%%%%%Plot IDM ---Position %%%%%%%%%%%%%%%
figure(2);
hold on;
builtin('plot',Tfinal,Xfinal,'k-','LineWidth', 1)
plot(Tfinal,X_f,'r-','Linewidth',1)

xlabel('Time (sec)')
ylabel('Position (m)')
xlim([0 900])
legend([builtin('plot',Tfinal,Xfinal,'k-','LineWidth', 1)...
    plot(Tfinal,X_f,'r-','Linewidth',1)], ...
    'Preceeding truck','Follower truck');
grid on
box on

%%%%%%%%%% Determine the follower truck's gaps %%%%%%%%%%%%
gap_idm = Xfinal - X_f - l;
Th_idm = gap_idm(:)./V_f(:);
```

```
%%%%%%%%%%%% Plot IDM ---Gap %%%%%%%%%%%%%%
figure(3);
hold on;
plot(Tfinal,gap_idm,'r-','Linewidth',1)

xlabel('Time (sec)')
ylabel('Inter-truck gaps (m)')

legend([plot(Tfinal,gap_idm,'r-','Linewidth',1)], ...
    'Preceeding truck & follower truck');
grid on;
box on;

%%%%%%%%%%%% Plot IDM ---Time gap %%%%%%%%%%%%%%
figure(4);
hold on;
plot(Tfinal,Th_idm,'r-','Linewidth',1)

xlabel('Time (sec)')
ylabel('Time gap (sec)')

legend([plot(Tfinal,Th_idm,'r-','Linewidth',1)], ...
    'Preceeding truck & follower truck');
```

```
grid on;

box on;


%%%%%%%%%%%% Save trajectory %%%%%%%%%%%%%%

writematrix(X_f, 'X_f1.txt');

writematrix(V_f, 'V_f1.txt');


# Codes for Plot

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%

close all; clear all; clc;


%%%%%%%%%%%% INPUT PARAMETERS  %%%%%%%%%%%%

N = 4; %Number of follower trucks

l = 15; %Length of trucks

F_time = 900;   % final time

Th_const = 0.5;


%%%%%%%%%%%% Load all trucks' trajectories %%%%%%%%%%%%%%

T = load('T.txt');

Vfinal = load('V_L.txt');

Xfinal = load('X_L.txt');

Xfinal1 = load('X_f1.txt');
```

```
Vfinal1 = load('V_f1.txt');

Xfinal2 = load('X_f2.txt');

Vfinal2 = load('V_f2.txt');

Xfinal3 = load('X_f3.txt');

Vfinal3 = load('V_f3.txt');

Xfinal4 = load('X_f4.txt');

Vfinal4 = load('V_f4.txt');


Yidm=[Xfinal1,Xfinal2,Xfinal3,Xfinal4,...

   Vfinal1,Vfinal2,Vfinal3,Vfinal4];

% ------------------------------------------------------------------

%%%%%%%%%%%Plot IDM ---Velocity%%%%%%%%%%%%%%

figure(1);

hold on;

builtin('plot',T,Vfinal,'k-','LineWidth', 1)

plot(T,Yidm(:,N+1),'r-','Linewidth',1)

plot(T,Yidm(:,N+2),'g--','Linewidth',1)

plot(T,Yidm(:,N+3),'m-.','Linewidth',1)

plot(T,Yidm(:,N+4),'b:','Linewidth',1.5)



xlabel('Time (s)')

ylabel('Speed (m/s)')
```

```
xlim([0 900])
ylim([15 35])
legend([builtin('plot',T,Vfinal,'k-','LineWidth', 1) ...
    plot(T,Yidm(:,N+1),'r-','Linewidth',1) ...
    plot(T,Yidm(:,N+2),'g--','Linewidth',1) ...
    plot(T,Yidm(:,N+3),'m-.','Linewidth',1) ...
    plot(T,Yidm(:,N+4),'b:','Linewidth',1.5)], ...
    'Leader truck','Follower truck 1','Follower truck 2',...
    'Follower truck 3','Follower truck 4');
grid on
box on

%%%%%%%%%%%%Plot IDM ---Position %%%%%%%%%%%%%%%
figure(2);
hold on
builtin('plot',T,Xfinal,'k-','LineWidth', 1)
plot(T,Yidm(:,1),'r-','Linewidth',1)
plot(T,Yidm(:,2),'g--','Linewidth',1)
plot(T,Yidm(:,3),'m-.','Linewidth',1)
plot(T,Yidm(:,4),'b:','Linewidth',1.5)


xlabel('Time (s)')
```

```
ylabel('Position (m)')

xlim([0 900])

legend([builtin('plot',T,Xfinal,'k-','LineWidth', 1)...

plot(T,Yidm(:,1),'r-','Linewidth',1)...

plot(T,Yidm(:,2),'g--','Linewidth',1)...

plot(T,Yidm(:,3),'m-.','Linewidth',1)...

plot(T,Yidm(:,4),'b:','Linewidth',1.5)], ...

    'Leader truck','Follower truck 1','Follower truck 2',...

    'Follower truck 3','Follower truck 4');

grid on

box on


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%


[r,c] = size(Yidm);

gap_idm = zeros(r,N);

Th_idm = zeros(r,N);


for i = 1:N

    if i==1

        gap_idm(:,i) = Xfinal - Yidm(:,i) - l;

    else
```

```
        gap_idm(:,i) = Yidm(:,i-1) - Yidm(:,i) - l;
    end
end

for i = 1:N
    Th_idm(:,i) = gap_idm(:,i)./Yidm(:,N+i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
% %%%%%%%%%%%%%%%Plot IDM ---Gap %%%%%%%%%%%%%%%%
figure(3);
hold on
plot(T,gap_idm(:,1),'r-','Linewidth',1)
plot(T,gap_idm(:,2),'g--','Linewidth',1)
plot(T,gap_idm(:,3),'m-.','Linewidth',1)
plot(T,gap_idm(:,4),'b:','Linewidth',1.5)



xlabel(['Time (s)'])
ylabel('Space gap with preceding truck (m)')
xlim([0 900])
```

```
legend([plot(T,gap_idm(:,1),'r-','Linewidth',1)...

plot(T,gap_idm(:,2),'g--','Linewidth',1)...

plot(T,gap_idm(:,3),'m-.','Linewidth',1)...

plot(T,gap_idm(:,4),'b:','Linewidth',1.5)], ...

    'Follower truck 1','Follower truck 2', ...

    'Follower truck 3','Follower truck 4');

grid on

box on

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%

% %%%%%%%%%%%%%Plot IDM ---Time gap %%%%%%%%%%%%%%%

figure(4);

hold on

plot(T,Th_idm(:,1),'r-','Linewidth',1)

plot(T,Th_idm(:,2),'g--','Linewidth',1)

plot(T,Th_idm(:,3),'m-.','Linewidth',1)

plot(T,Th_idm(:,4),'b:','Linewidth',1.5)



xlabel(['Time (s)'])

ylabel('Time gap with preceding truck (s)')

xlim([0 900])
```

```
legend([plot(T,Th_idm(:,1),'r-','Linewidth',1)...
plot(T,Th_idm(:,2),'g--','Linewidth',1)...
plot(T,Th_idm(:,3),'m-.','Linewidth',1)...
plot(T,Th_idm(:,4),'b:','Linewidth',1.5)], ...
    'Follower truck 1','Follower truck 2', ...
    'Follower truck 3','Follower truck 4');
grid on
box on
%%%%%%%%%%%%%%%Plot ACC & JERK%%%%%%%
[r,c] = size(Yidm);
acc_idm = zeros(r, N-1);
jerk_idm = zeros(r, N-1);

del_t = T(2) - T(1);

for j = 1:N
    for i = 1:r-1
            acc_idm(i,j) = (Yidm(i+1,N+j) - Yidm(i,N+j))/del_t;
    end
end

for j = 1:N
    for i = 1:r-1
```

```
        jerk_idm(i,j) = (acc_idm(i+1,j) - acc_idm(i,j))/del_t;
    end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%Plot IDM ---Acc %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(5);
hold on
plot(T,acc_idm(:,1),'r-','Linewidth',1)
plot(T,acc_idm(:,2),'g--','Linewidth',1)
plot(T,acc_idm(:,3),'m-.','Linewidth',1)
plot(T,acc_idm(:,4),'b:','Linewidth',1.5)

xlabel(['Time (s)'])
ylabel('Acceleration (m/s^2)')
xlim([0 900])

legend([plot(T,acc_idm(:,1),'r-','Linewidth',1)...
plot(T,acc_idm(:,2),'g--','Linewidth',1)...
plot(T,acc_idm(:,3),'m-.','Linewidth',1)...
plot(T,acc_idm(:,4),'b:','Linewidth',1.5)], ...
    'Follower truck 1','Follower truck 2', ...
```

```
    'Follower truck 3','Follower truck 4');
grid on
box on

%%%%%%%%%%%%Plot IDM ---Jerk %%%%%%%%%%%%%%%%
figure(6);
hold on
plot(T,jerk_idm(:,1),'r-','Linewidth',1)
plot(T,jerk_idm(:,2),'g--','Linewidth',1)
plot(T,jerk_idm(:,3),'m-.','Linewidth',1)
plot(T,jerk_idm(:,4),'b:','Linewidth',1.5)

xlabel(['Time (s)'])
ylabel('Jerk (m/s^3)')
xlim([0 900])

legend([plot(T,jerk_idm(:,1),'r-','Linewidth',1)...
plot(T,jerk_idm(:,2),'g--','Linewidth',1)...
plot(T,jerk_idm(:,3),'m-.','Linewidth',1)...
plot(T,jerk_idm(:,4),'b:','Linewidth',1.5)], ...
    'Follower truck 1','Follower truck 2', ...
    'Follower truck 3','Follower truck 4');
grid on
```

```
box on


% %%%%%%%%%%%%%%%%%%%%%%% Plot SSSE
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
E_vel_idm = [Vfinal, Yidm(:,N+1:end-1)] - Yidm(:,N+1:end);
SSE_vel_idm  = sum(E_vel_idm.^2, 2);


figure(7);
hold on;
plot(T,SSE_vel_idm,'g-.','LineWidth',2)
xlabel('Time (sec)')
xlim([0 900])
ylabel('SSSE (m^2/sec^2)')
% legend([plot(T,SSE_vel_idm,'g-.','LineWidth',2)], ...
%     '#####');
grid on
box on
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
set(findall(gcf,'-property','FontSize'),'FontSize',12)
```

Appendix B

Python Code for Trajectory Prediction

```
# Defining LSTM and QLSTM Model

import torch

from torch import nn

from torch.utils.data import Dataset

import pennylane as qml


class SequenceDataset(Dataset):

def __init__(self, dataframe, target, features, sequence_length=5):

self.features = features

self.target = target

self.sequence_length = sequence_length

self.y = torch.tensor(dataframe[self.target].values).float()

self.X = torch.tensor(dataframe[self.features].values).float()


def __len__(self):

return self.X.shape[0]


def __getitem__(self, i):

if i >= self.sequence_length - 1:

i_start = i - self.sequence_length + 1

x = self.X[i_start:(i + 1), :]
```

```
else:
padding = self.X[0].repeat(self.sequence_length - i - 1, 1)
x = self.X[0:(i + 1), :]
x = torch.cat((padding, x), 0)

return x, self.y[i]

class ShallowRegressionLSTM(nn.Module):
def __init__(self, num_sensors, hidden_units):
super().__init__()
self.num_sensors = num_sensors # this is the number of features
self.hidden_units = hidden_units
self.num_layers = 1

self.lstm = nn.LSTM(
input_size=num_sensors,
hidden_size=hidden_units,
batch_first=True,
num_layers=self.num_layers
)

self.linear = nn.Linear(in_features=self.hidden_units, out_features=1)
```

```
def forward(self, x):

batch_size = x.shape[0]

h0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).requires_grad_()

c0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).requires_grad_()

_, (hn, _) = self.lstm(x, (h0, c0))

out = self.linear(hn[0]).flatten() # First dim of Hn is num_layers, which is set to 1 above.


return out

class QLSTM(nn.Module):

def __init__(self,

input_size,

hidden_size,

n_qubits=4,

n_qlayers=1,

n_vrotations=3,

batch_first=True,

return_sequences=False,

return_state=False,

backend="default.qubit"):

super(QLSTM, self).__init__()

self.n_inputs = input_size

self.hidden_size = hidden_size

self.concat_size = self.n_inputs + self.hidden_size
```

```
self.n_qubits = n_qubits
self.n_qlayers = n_qlayers
self.n_vrotations = n_vrotations
self.backend = backend # "default.qubit", "qiskit.basicaer", "qiskit.ibm"

self.batch_first = batch_first
self.return_sequences = return_sequences
self.return_state = return_state
self.wires_forget = [f"wire_forget_{i}" for i in range(self.n_qubits)]
self.wires_input = [f"wire_input_{i}" for i in range(self.n_qubits)]
self.wires_update = [f"wire_update_{i}" for i in range(self.n_qubits)]
self.wires_output = [f"wire_output_{i}" for i in range(self.n_qubits)]

self.dev_forget = qml.device(self.backend, wires=self.wires_forget)
self.dev_input = qml.device(self.backend, wires=self.wires_input)
self.dev_update = qml.device(self.backend, wires=self.wires_update)
self.dev_output = qml.device(self.backend, wires=self.wires_output)

#self.dev_forget = qml.device(self.backend, wires=self.n_qubits)
#self.dev_input = qml.device(self.backend, wires=self.n_qubits)
#self.dev_update = qml.device(self.backend, wires=self.n_qubits)
#self.dev_output = qml.device(self.backend, wires=self.n_qubits)
def ansatz(params, wires_type):
```

```
# Entangling layer.
for i in range(1,3):
for j in range(self.n_qubits):
if j + i < self.n_qubits:
qml.CNOT(wires=[wires_type[j], wires_type[j + i]])
else:
qml.CNOT(wires=[wires_type[j], wires_type[j + i - self.n_qubits]])

# Variational layer.
for i in range(self.n_qubits):
qml.RX(params[0][i], wires=wires_type[i])
qml.RY(params[1][i], wires=wires_type[i])
qml.RZ(params[2][i], wires=wires_type[i])
def VQC(features, weights, wires_type):
# Preproccess input data to encode the initial state.
#qml.templates.AngleEmbedding(features, wires=wires_type)
ry_params = [torch.arctan(feature) for feature in features]
rz_params = [torch.arctan(feature**2) for feature in features]
for i in range(self.n_qubits):
qml.Hadamard(wires=wires_type[i])
qml.RY(ry_params[i], wires=wires_type[i])
qml.RZ(ry_params[i], wires=wires_type[i])
#Variational block.
```

```
qml.layer(ansatz, self.n_qlayers, weights, wires_type = wires_type)

def _circuit_forget(inputs, weights):
VQC(inputs, weights, self.wires_forget)
return [qml.expval(qml.PauliZ(wires=i)) for i in self.wires_forget]
self.qlayer_forget = qml.QNode(_circuit_forget, self.dev_forget, interface="torch")

def _circuit_input(inputs, weights):
VQC(inputs, weights, self.wires_input)
return [qml.expval(qml.PauliZ(wires=i)) for i in self.wires_input]
self.qlayer_input = qml.QNode(_circuit_input, self.dev_input, interface="torch")

def _circuit_update(inputs, weights):
VQC(inputs, weights, self.wires_update)
return [qml.expval(qml.PauliZ(wires=i)) for i in self.wires_update]
self.qlayer_update = qml.QNode(_circuit_update, self.dev_update, interface="torch")

def _circuit_output(inputs, weights):
VQC(inputs, weights, self.wires_output)
return [qml.expval(qml.PauliZ(wires=i)) for i in self.wires_output]
self.qlayer_output = qml.QNode(_circuit_output, self.dev_output, interface="torch")

weight_shapes = {"weights": (self.n_qlayers, self.n_vrotations, self.n_qubits)}
```

```
print(f"weight_shapes = (n_qlayers, n_vrotations, n_qubits) = ({self.n_qlayers},
{self.n_vrotations}, {self.n_qubits})")

self.clayer_in = torch.nn.Linear(self.concat_size, self.n_qubits)
self.VQC = {
'forget': qml.qnn.TorchLayer(self.qlayer_forget, weight_shapes),
'input': qml.qnn.TorchLayer(self.qlayer_input, weight_shapes),
'update': qml.qnn.TorchLayer(self.qlayer_update, weight_shapes),
'output': qml.qnn.TorchLayer(self.qlayer_output, weight_shapes)
}
self.clayer_out = torch.nn.Linear(self.n_qubits, self.hidden_size)
#self.clayer_out = [torch.nn.Linear(n_qubits, self.hidden_size) for _ in range(4)]

def forward(self, x, init_states=None):
'''
x.shape is (batch_size, seq_length, feature_size)
recurrent_activation -> sigmoid
activation -> tanh
'''
if self.batch_first is True:
batch_size, seq_length, features_size = x.size()
else:
seq_length, batch_size, features_size = x.size()
```

```
hidden_seq = []
if init_states is None:
h_t = torch.zeros(batch_size, self.hidden_size) # hidden state (output)
c_t = torch.zeros(batch_size, self.hidden_size) # cell state
else:
# for now we ignore the fact that in PyTorch you can stack multiple RNNs
# so we take only the first elements of the init_states tuple init_states[0][0], init_states[1][0]
h_t, c_t = init_states
h_t = h_t[0]
c_t = c_t[0]

for t in range(seq_length):
# get features from the t-th element in seq, for all entries in the batch
x_t = x[:, t, :]
# Concatenate input and hidden state
v_t = torch.cat((h_t, x_t), dim=1)

# match qubit dimension
y_t = self.clayer_in(v_t)

f_t = torch.sigmoid(self.clayer_out(self.VQC['forget'](y_t))) # forget block
i_t = torch.sigmoid(self.clayer_out(self.VQC['input'](y_t))) # input block
```

```
g_t = torch.tanh(self.clayer_out(self.VQC['update'](y_t))) # update block
o_t = torch.sigmoid(self.clayer_out(self.VQC['output'](y_t))) # output block

c_t = (f_t * c_t) + (i_t * g_t)
h_t = o_t * torch.tanh(c_t)

hidden_seq.append(h_t.unsqueeze(0))
hidden_seq = torch.cat(hidden_seq, dim=0)
hidden_seq = hidden_seq.transpose(0, 1).contiguous()
return hidden_seq, (h_t, c_t)
class QShallowRegressionLSTM(nn.Module):
def __init__(self, num_sensors, hidden_units, n_qubits=0, n_qlayers=1):
super().__init__()
self.num_sensors = num_sensors # this is the number of features
self.hidden_units = hidden_units
self.num_layers = 1

#self.lstm = nn.LSTM(
# input_size=num_sensors,
# hidden_size=hidden_units,
# batch_first=True,
# num_layers=self.num_layers
#)
```

```
self.lstm = QLSTM(
input_size=num_sensors,
hidden_size=hidden_units,
batch_first=True,
n_qubits = n_qubits,
n_qlayers= n_qlayers
)

self.linear = nn.Linear(in_features=self.hidden_units, out_features=1)

def forward(self, x):
batch_size = x.shape[0]
h0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).requires_grad_()
c0 = torch.zeros(self.num_layers, batch_size, self.hidden_units).requires_grad_()
_, (hn, _) = self.lstm(x, (h0, c0))
out = self.linear(hn).flatten() # First dim of Hn is num_layers, which is set to 1 above.

return out

# Importing Libraries
import helper
import pandas as pd
from utils import *
```

```
import time
import numpy as np
import math
import matplotlib.pyplot as plt
import torch
from torch.utils.data import DataLoader
from torch import nn
from IPython.display import Image
import pandas as pd
import seaborn as sns

#Importing Data Set
df = pd.read_csv(‘Train_Data.csv')
df
target = "Speed"
features = ['Time', 'X_Pos', 'Distance']

# Data Processing
size = int(len(df) * 0.70)
df_train = df.loc[:size].copy()
df_test = df.loc[size:].copy()
df_eval = pd.read_csv('Eval_Data.csv')
target_mean = df_train[target].mean()
```

```
target_stdev = df_train[target].std()

for c in df_train.columns:
    mean = df_train[c].mean()
    stdev = df_train[c].std()

    df_train[c] = (df_train[c] - mean) / stdev
    df_test[c] = (df_test[c] - mean) / stdev
    df_eval[c] = (df_eval[c] - mean) / stdev
from Factory import SequenceDataset

torch.manual_seed(101)

batch_size = 1
sequence_length = 3

train_dataset = SequenceDataset(
    df_train,
    target=target,
    features=features,
    sequence_length=sequence_length
)
test_dataset = SequenceDataset(
```

```
    df_test,
    target=target,
    features=features,
    sequence_length=sequence_length
)
eval_dataset = SequenceDataset(
    df_eval,
    target=target,
    features=features,
    sequence_length=sequence_length
)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
eval_loader = DataLoader(eval_dataset, batch_size=batch_size, shuffle=False)

X, y = next(iter(train_loader))

print("Features shape:", X.shape)
print("Target shape:", y.shape)

def train_model(data_loader, model, loss_function, optimizer):
    num_batches = len(data_loader)
    total_loss = 0
```

```
    model.train()

    for X, y in data_loader:
        output = model(X)
        loss = loss_function(output, y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    avg_loss = total_loss / num_batches
    print(f"Train loss: {avg_loss}")
    return avg_loss

def test_model(data_loader, model, loss_function):

    num_batches = len(data_loader)
    total_loss = 0

    model.eval()
    with torch.no_grad():
```

```
        for X, y in data_loader:
            output = model(X)
            total_loss += loss_function(output, y).item()

    avg_loss = total_loss / num_batches
    print(f"Test loss: {avg_loss}")
    return avg_loss

def eval_model(data_loader, model, loss_function):

    num_batches = len(data_loader)
    total_loss = 0

    model.eval()
    with torch.no_grad():
        for X, y in data_loader:
            output = model(X)
            total_loss += loss_function(output, y).item()

    avg_loss = total_loss / num_batches
    print(f"Test loss: {avg_loss}")
    return avg_loss
```

```
# Running the Classical LSTM
from Factory import ShallowRegressionLSTM
learning_rate = 0.0001
num_hidden_units = 16

model = ShallowRegressionLSTM(num_sensors=len(features),
hidden_units=num_hidden_units)
loss_function = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
classical_loss_train = []
classical_loss_test = []
print("Untrained test\n--------")
test_loss = test_model(test_loader, model, loss_function)
print()
classical_loss_test.append(test_loss)

for ix_epoch in range(20):
    print(f"Epoch {ix_epoch}\n---------")
    train_loss = train_model(train_loader, model, loss_function, optimizer=optimizer)
    test_loss = test_model(test_loader, model, loss_function)
    print()
    classical_loss_train.append(train_loss)
    classical_loss_test.append(test_loss)
```

```
def predict(data_loader, model):
    """Just like `test_loop` function but keep track of the outputs instead of the loss
    function.
    """
    output = torch.tensor([])
    model.eval()
    with torch.no_grad():
        for X, _ in data_loader:
            y_star = model(X)
            output = torch.cat((output, y_star), 0)

    return output
train_eval_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False)
ystar_col = "Model forecast"
df_train[ystar_col] = predict(train_eval_loader, model).numpy()
df_test[ystar_col] = predict(test_loader, model).numpy()
df_out = pd.concat((df_train, df_test))[[target, ystar_col]]

for c in df_out.columns:
    df_out[c] = df_out[c] * target_stdev + target_mean

print(df_out)
df_eval[ystar_col] = predict(eval_loader, model).numpy()
```

```
df_out1 = pd.concat((df_train, df_eval))[[target, ystar_col]]
for c in df_out1.columns:
    df_out1[c] = df_out1[c] * target_stdev + target_mean

print(file_name = 'Leader_Prediction_LSTM.xlsx'
df_out1.to_excel(file_name)df_out1)

# Running the QLSTM
from Factory import QShallowRegressionLSTM
learning_rate = 0.05
num_hidden_units = 16

Qmodel = QShallowRegressionLSTM(num_sensors=len(features),
hidden_units=num_hidden_units, n_qubits=4)
loss_function = nn.MSELoss()
optimizer = torch.optim.Adagrad(Qmodel.parameters(), lr=learning_rate)
quantum_loss_train = []
quantum_loss_test = []
print("Untrained test\n--------")
start = time.time()
test_loss = test_model(test_loader, Qmodel, loss_function)
end = time.time()
print("Execution time", end - start)
```

```
quantum_loss_test.append(test_loss)


for ix_epoch in range(20):
    print(f"Epoch {ix_epoch}\n---------")
    start = time.time()
    train_loss = train_model(train_loader, Qmodel, loss_function, optimizer=optimizer)
    test_loss = test_model(test_loader, Qmodel, loss_function)
    end = time.time()
    print("Execution time", end - start)
    quantum_loss_train.append(train_loss)
    quantum_loss_test.append(test_loss)
train_eval_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=False)


ystar_col_Q = "Model forecast"
df_train[ystar_col_Q] = predict(train_eval_loader, Qmodel).numpy()
df_test[ystar_col_Q] = predict(test_loader, Qmodel).numpy()


df_out_Q = pd.concat((df_train, df_test))[[target, ystar_col_Q]]


for c in df_out_Q.columns:
    df_out_Q[c] = df_out_Q[c] * target_stdev + target_mean


print(df_out_Q)
```

```
df_eval[ystar_col_Q] = predict(eval_loader, model).numpy()
df_out1_Q = pd.concat((df_train, df_eval))[[target, ystar_col]]
for c in df_out1_Q.columns:
    df_out1_Q[c] = df_out1_Q[c] * target_stdev + target_mean

print(df_out1_Q)
file_name1 = 'Leader_Prediction_QLSTM.xlsx'
df_out1.to_excel(file_name1)

# Comparison of Train Loss Values
plt.figure(figsize=(8, 6))
plt.plot(classical_loss_train, label = "LSTM")
plt.plot(quantum_loss_train, label = "QLSTM")
plt.ylabel('Train Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()

 # Comparison of Test Loss Values
plt.figure(figsize=(8, 6))
plt.plot(classical_loss_test, label = "LSTM")
plt.plot(quantum_loss_test, label = "QLSTM")
plt.ylabel('Test Loss')
```

```
plt.xlabel('Epoch')
plt.legend()
plt.show()

# Comparison of Number of Parameters Used
total_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
total_params_Q = sum(p.numel() for p in Qmodel.parameters() if p.requires_grad)

print("No. of parameters for Classical LSTM: ", total_params)
print("No. of parameters for QLSTM: ", total_params_Q)
```

REFERENCES

Abbas, A., Sutter, D., Zoufal, C., Lucchi, A., Figalli, A., & Woerner, S., 2021. The power of quantum neural networks. *[Nature Computational Science,* 403–409.

Alam, A., Besselink, B., Turri, V., Martensson, J., Johansson, K.H., 2015. Heavy-duty vehicle platooning for sustainable freight transportation: a cooperative method to enhance safety and efficiency IEEE Control Syst, 35, pp. 34-56, 10.1109/MCS.2015.2471046

Al-Qadi, I.L., Okte, E., Ramakrishnan, A., Zhou, Q. and Sayeh, W., 2021. Truck-Platoonable Pavement Sections in Illinois' Network. Illinois Center for Transportation/Illinois Department of Transportation

Azad, U., Behera, B. K., Ahmed, E. A., Panigrahi, P. K., & Farouk, A., 2022. Solving Vehicle Routing Problem Using Quantum Approximate Optimization Algorithm. *IEEE Transactions on Intelligent Transportation Systems,* 1-10.

Bassman, L., Urbanek, M., Metcalf, M., Carter, J., Kemper, A. F., & Jong, W. A., 2021. Simulating quantum materials with digital quantum computers. *Quantum Science and Technology, Volume 6, Number 4*, 043002.

Brackstone, M., and McDonald, M. "Car-Following: A Historical Review". Transportation Research Part F: Traffic Psychology and Behavior, Vol. 2, No. 4, pp. 181-196, 1999.

Busemeyer, J. R., & Bruza, P. D., 2012. *Quantum models of cognition and decision.* Cambridge University Press.

Chen, H., Miao, F., & Shen, X., 2020. Hyperspectral Remote Sensing Image Classification with CNN Based on Quantum Genetic-Optimized Sparse Representation. *IEEE Access (Volume: 8)*, 99900 – 99909.

Chen, S.Y., Yoo, S., & Fang, Y.L., 2020. Quantum Long Short-Term Memory, https://arxiv.org/pdf/2009.01783.pdf.

Crosson, E., & Harrow, A. W., 2016. Simulated Quantum Annealing Can Be Exponentially Faster Than Classical Simulated Annealing. *IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)* (p. DOI: 10.1109/FOCS.2016.81). New Brunswick, NJ, USA: IEEE.

Dai, S., Li, L., & Z. Li, "Modeling vehicle interactions via modified lstm models for trajectory prediction", *IEEE Access*, vol. 7, pp. 38 287-38 296, 2019

Dou, Y., Yan, F., and Feng, D., 2016. "Lane changing prediction at highway lane drops using support vector machine and artificial neural network classifiers", pp. 901-906.

Du, H., Leng, S., He, J., & Zhou, L., 2021. "Digital Twin Based Trajectory Prediction for Platoons of Connected Intelligent Vehicles," *IEEE 29th International Conference on Network Protocols (ICNP)*, Dallas, TX, USA, 2021, pp. 1-6, doi: 10.1109/ICNP52444.2021.9651970

Dunjko, V. and Briegel, H.J., 2018. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. Reports on Progress in Physics, 81(7), p.074001.

Dunjko, V., & Wittek, P., 2020. A non-review of Quantum Machine Learning: Trends and exploration. *The open journal for quantum science, Volume 4*, 32.

EAMA, “Infographic on EU roadmap for truck platooning,” 2019,. https://www.acea.be/uploads/publications/Platooning_roadmap.pdf.

Economics and Industry Data [WWW document], 2022. URL https://www.trucking.org/economics-and-industry-data (accessed 2.20.22)

Hochreiter S, Schmidhuber J. Long short-term memory. Neural computation. 1997;9(8):1735–1780. doi: 10.1162/neco.1997.9.8.1735

Islam, M., Chowdhury, M., Khan, Z., and Khan, S. M., 2022. "Hybrid Quantum-Classical Neural Network for Cloud-Supported In-Vehicle Cyberattack Detection," in IEEE Sensors Letters, vol. 6, no. 4, pp. 1-4, Art no. 6001204, doi: 10.1109/LSENS.2022.3153931

Jeong, E., Oh, C., & Lee, S., 2017. "Is vehicle automation enough to prevent crashes? Role of traffic operations in automated driving environments for traffic safety," *Accident Analysis & Prevention*, vol. 104, pp. 115–124.

Khan, Z., Khan, S. M., Tine, J. M., Comert, A. T., Rice, D., Comert, G., . . . Chowdhury, M., 2021. Hybrid Quantum-Classical Neural Network for Incident Detection. *DeepAI*, https://arxiv.org/ftp/arxiv/papers/2108/2108.01127.pdf.

Lammert, M.P., Bugbee, B., Hou, Y., Muratori, M., Holden, J., Duran, A.W., Mack, A. and Swaney, E., 2018. Exploring telematics big data for truck platooning opportunities (No. NREL/CP-5400-70869). National Renewable Energy Lab. (NREL), Golden, CO (United States)

Lee, S., Oh, C., Lee, G., "Impact of Automated Truck Platooning on the Performance of Freeway Mixed Traffic Flow", *Journal of Advanced Transportation*, vol. 2021, Article ID 8888930, 13 pages, 2021. https://doi.org/10.1155/2021/8888930.

Ma, X., Huo, E., Yu, H., Li, H., 2021. "Mining truck platooning patterns through massive trajectory data", Knowledge-Based Systems, Volume 221, 106972, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2021.106972.

Noruzoliaee, M., Zou, B. and Zhou, Y.J., 2021. Truck platooning in the US national road network: A system-level modeling approach. Transportation Research Part E: Logistics and Transportation Review, 145, p.102200.

Panwai, S., and Dia, H., 2005. "Comparative Evaluation of Microscopic Car-Following Behavior". IEEE Transactions on Intelligent Transportation Systems, Vol. 6, No. 3, pp. 314-325.

Papadoulis, A., Quddus, M., & and Imprialou, M., 2019. "Evaluating the safety impact of connected and autonomous vehicles on motorways," *Accident Analysis & Prevention*, vol. 124, pp. 12–22.

Patel, O. P., & Tiwari, A., 2014. Quantum Inspired Binary Neural Network Algorithm. *International Conference on Information Technology* (p. DOI: 10.1109/ICIT.2014.29). Bhubaneswar, India: IEEE.

Patel, O. P., Tiwari, A., Chaudhary, R., Nuthalapati, S. V., Bharill, N., Prasad, M., . . . Hussain, O. K., 2019. Enhanced quantum-based neural network learning and its application to signature verification. *Soft Computing, Volume 23*, 3067–3080.

Rahman, M. S. and Abdel-Aty, M., 2018. "Longitudinal safety evaluation of connected vehicles' platooning on expressways," *Accident Analysis & Prevention*, vol. 117, pp. 381–391.

Rahman, M., Chowdhury, M., Dey, K., Islam, M.R., Khan, T., 2017. Evaluation of Driver Car-Following Behavior Models for Cooperative Adaptive Cruise Control Systems. Transportation Research Record 2622, 84–95. https://doi.org/10.3141/2622-08

Rahman, M., Khan, S.M., Chowdhury, M., Huynh, N., Ogle, J., Dey, K., Bhavsar, P., 2015. Incident Command System Strategies for Incident Management on Freeways: A Simulation Analysis. Presented at the Transportation Research Board 94th Annual Meeting Transportation Research Board

Salek, M Sabbir, "Theoretical Development and Numerical Validation of an Asymmetric Linear Bilateral Control Model for an Automated Truck Platoon" (2021). All Theses. 3708

Sgarbas, K. N., 2007. The Road to Quantum Artificial Intelligence. *Wire Communications Lab., Dept. of Electrical and Computer Engineering*, arXiv preprint arXiv:0705.3360.

Shladover, S.E., Yun, X., Yang, L., Ramezani, H., Spring, J., Nowakowski, C.V., Nelson, D., Thompson, D. and Kailas, A., 2018. Cooperative adaptive cruise control (CACC) for partially automated truck platooning (No. CA18-2623). California. Dept. of Transportation. Division of Research and Innovation.

Song, Q., Fu, W., Wang, W., Sun, Y., Wang, D., & Zhou, J., 2022. Quantum decision making in automatic driving. *Scientific reports 12.1*, 1-15.

Song, Q., Wang, W., Fu, W., Sun, Y., Wang, D., & Gao, Z., 2022. Research on quantum cognition in autonomous driving. *Scientific Reports Volume 12, Article number: 300*, https://doi.org/10.1038/s41598-021-04239-y.

Treiber, M., Hennecke, A., & Helbing, D., 2000. Congested Traffic States in Empirical Observations and Microscopic Simulations. Physical Review E, Vol. 62, No. 2, 2000, pp. 1805–1824. https://doi.org/10.1103/PhysRevE.62.1805.

Tsugawa, S., Jeschke, S., Shladover, S.E., 2016. A Review of Truck Platooning Projects for Energy Savings. IEEE Transactions on Intelligent Vehicles 1, 68–77. https://doi.org/10.1109/TIV.2016.2577499

Wang, L.-J., Lin, J.-Y., & Wu, S., 2022. Implementation of quantum stochastic walks for function approximation, two-dimensional data classification, and sequence classification. *Quantum Physics*, arXiv:2103.03018.

Wang, M., Maarseveen, S., Happee, R., Tool, O., & Arem, B.V, 2019. "Benefits and risks of truck platooning on freeway operations near entrance ramp," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2673, no. 8, pp. 588–602.

Wei, J., He, J., Zhou, Y., Chen, K., Tang, Z., & Xiong, Z., 2020. "Enhanced object detection with deep convolutional neural networks for advanced driving assistance", *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 4, pp. 1572-1583.

Wiebe, N., Kapoor, A., & Svore, K. M., 2014. Quantum Deep Learning. *Quantum Physics*, arXiv:1412.3489.

Yan L., & Shen H., 2019. TOP: optimizing vehicle driving speed with vehicle trajectories for travel time minimization and road congestion avoidance. ACM Transactions on Cyber-Physical Systems, 4(2), 1–25. doi: 10.1145/336216

Zhang, B., Wilschut, E. S., Willemsen, D. M. C., & Martens, M. H., 2019. “Transitions to manual control from highly automated driving in non-critical truck platooning scenarios,” *Transportation Research Part F: Traffic Psychology and Behaviour*, vol. 64, pp. 84–97.

Zhang, D., Wang, J., Fan, H., Zhang, T., Gao, J., & Yang, P., 2020. New method of traffic flow forecasting based on quantum particle swarm optimization strategy for intelligent transportation system. International Journal of Communication Systems, https://doi.org/10.1002/dac.4647.