Abstract—Quantum computing (QC) has received a lot of attention according to its light training parameter numbers and computational speeds by qubits. Moreover, various researchers have tried to enable quantum machine learning (QML) using QC, where there are also multifarious efforts to use QC to implement quantum multi-agent reinforcement learning (QMARL). Existing classical multi-agent reinforcement learning (MARL) using neural network features non-stationarity and uncertain properties due to its large number of parameters. Therefore, this paper presents a visual simulation software framework for a novel QMARL algorithm to control autonomous multi-drone systems to take advantage of QC. Our proposed QMARL framework accomplishes reasonable reward convergence and service quality performance with fewer trainable parameters than the classical MARL. Furthermore, QMARL shows more stable training results than existing MARL algorithms. Lastly, our proposed visual simulation software allows us to analyze the agents’ training process and results.

I. INTRODUCTION

Spurred by the recent advances in QC, attempts to re-implement existing machine learning (ML) have been presented to discover quantum advantages [1]. Primarily, MARL is one of the most challenging ML fields due to scalability and non-stationarity [2]. Yun et al. have proposed QMARL that can perform reasonably by using a small number of trainable parameters compared to classical neural network [3]. In addition, meta QMARL has been proposed, which enables adaptation to context changes by using a tiny size of memory (known as pole memory) compared to the classical method [2]. Although these studies have shown the effectiveness and feasibility, user-friendly simulation software is not considered and designed yet.

For existing classical RL, OpenAI has already developed and publicized Gym [4], which helps compare the performance between various RL algorithms by implementing communication between the algorithms and the environments. Furthermore, it has become standard to evaluate RL algorithms via Gym environments. Many versions of Gym exist for different frameworks of RL. For example, Gazebo is a library for 3D robotics simulation in which RL can be utilized. Petting Zoo [5] and Starcraft Multi-agent challenge (SMAC) [6] are both environments for MARL equipped with multiple agents. However, SMAC specializes in decentralized micromanagement scenarios.

Next, for QML frameworks, a variety of tools such as TorchQuantum [7], and Qiskit [8] has also been developed and publicized in recent years. These libraries significantly contributed to the research of many current QML algorithms. For example, the QMARL framework mentioned above [2] also utilizes TorchQuantum for the implementation.

Similar to the frameworks introduced above, we have designed a QMARL software framework for visualizing a multi-drone communication environment. The MARL framework proposed in this paper is a hybrid model which utilizes both classic and QC. Classical computing is used to implement the Trainer in Fig. [1] which computes the loss function, performs optimization via gradient descent, and updates the target values. QC is used to implement the quantum-based policy (Q-policy) in Q-policy Layer, which computes the action distribution of all the agents. In this process, the data re-uploading technique encodes both classical data and compute output data. Leveraging QC in our proposed model has not only increased the computational speed exponentially but also shown a higher total reward value after training. Therefore, the proposed model outperforms the previous model and other benchmarks, corroborating our methodology’s efficiency. Moreover, we can analyze multi-agent training processes and results thanks to our proposed visual simulation software.

Contributions. The contributions of this work are as follows.

- A proposed QMARL framework for a multi-drone environment is designed to maximize users’ support rate and quality of service (QoS).
- A Q-policy network that computes agents’ action distribution via data re-uploading is proposed.

II. BASICS OF QUANTUM COMPUTING

Qubits. In QC, a qubit is used as the basic unit of information instead of bits. While bits can only be expressed by 0 or 1, qubits can be expressed as a combination of two bases, \(|0\rangle\) and \(|1\rangle\). This nature is also known as superposition. Furthermore, quantum states can be expressed by using qubits [9], and entanglement can occur between qubits which causes individual qubits to be strongly correlated. Due to these differences, qubits can express more information than classical bits. Assuming there is a qubits system, the quantum state defined within the Hilbert state can be defined as follows, 

\[ |\psi\rangle = \alpha_1 |0\cdots0\rangle + \cdots + \alpha_{2^n} |1\cdots1\rangle, \]

where \(\alpha\) represents the
probability amplitude and \( \sum_{i=1}^{2^k} a_i^2 = 1 \); finally, the quantum state can be graphically expressed on the Bloch sphere.

**Basic Quantum Gates.** Although bits and qubits are different, classical data can be encoded into qubits via quantum state encoders composed of basic quantum rotation gates expressed as \( R_x, R_y, \) and \( R_z \). These rotation gates encode classical bits into qubits and perform unitary operations on a qubit by rotating it in the direction of \( x-, y- \), and \( z- \) axes, respectively. Additionally, as mentioned above, qubit entanglement between two qubits can be achieved by a \( CNOT \) gate \[^{[10]}\]. This is done by performing the \( XOR \) operation on one qubit and using the other as the control qubit. All the gates introduced above are used to compose quantum neural networks.

**Quantum Neural Networks.** The structure of quantum neural networks (QNN) comprises the state encoder, parametrized quantum circuit (PQC), and the measurement layer. Since classical data \( X \) is incompatible with quantum circuits, it must be converted into quantum states before being used as input data. In this paper, the data re-uploading \[^{[11]}\] method will be used in implementing the state encoder as shown in Fig. \[^{[1]}\] Data reuploading and quantum state encoding are jointly achieved by passing \( q \) number of \( \{0\} \) qubits through a sequence of unitary operation gates containing the information of \( X \) and trainable parameters of the encoder, denoted as \( \theta_{enc} \). In order to imbue each gate with information of \( X \), it has to be split into \( [x_1 \cdots x_N] \). Then, these gates will repeatedly rotate the initial qubits, and the output quantum states produced successfully convert \( X \) into quantum states. The process of state encoding can be expressed as follows,\(^1\)

\[
|\psi_{enc}\rangle = U(\theta_N)U(x_N) \cdots U(\theta_1)U(x_1)|\psi_0\rangle, \tag{1}
\]

where \( N \) is the number of split data, \( U(\cdot) \) represents a unitary operation of qubit rotation, \( |\psi_0\rangle \) is the initial quantum state, \( |\psi_{enc}\rangle \) is the encoded qubit and \( \theta \) is the trainable parameters of the state encoder. The PQC will then process the converted qubits. As shown in \[^{[1]}\] the PQC is composed of the \( R_x, R_y, R_z \) and the \( CNOT \) gates which contain trainable parameters \( \theta_{PQC} \). By rotating and entangling the input qubits, we will be able to obtain the values needed for MARL. This can be expressed as \( |\psi_{PQC}\rangle = U(\theta_{PQC})|\psi_{enc}\rangle \). Finally, the produced qubits will proceed into the measurement layer, where they will be measured. Measurement can be carried out by applying a projection matrix \( M \in \mathbb{C} \equiv \{M_1, \cdots, M_c, \cdots, M_C\} \) onto the \( z- \) axis. The value produced from this operation is known as observable, denoted as \( \langle V \rangle_\theta \in [-1, 1]^C \). The measurement operation can be expressed as the following equation,\(^2\)

\[
\langle V \rangle_\theta = \langle 0|U^1(\theta)M_iU(\theta)U(x)|0 \rangle = \langle \psi|M_i|\psi \rangle, \tag{2}
\]

where \( (\cdot)^\dagger \) represents the complex conjugate operator. The observable produced from the quantum layer is the action distribution of the actors and is used to calculate the loss function. However, the loss function cannot be trained by backpropagation because quantum states within the QNN cannot be measured and will collapse when the chain rule is applied. Therefore, training is carried out by computing the loss gradient from the symmetric difference quotient of the loss function.

### III. SOFTWARE TOOL DESCRIPTION

**Framework of Proposed QMARL.** Fig. \[^{[1]}\] illustrates our overall proposed framework of quantum MARL. We consider the MARL environment in \[^{[12]}\], where agents try to learn a
policy that maximizes the total reward. We will describe the sequential framework of our proposed quantum MARL based on Fig. 1 (i) In Drone Environment Layer [12], all M agents collect observation information denoted as the set of states, which is \( S \equiv \{ o_1, \cdots , o_m, \cdots , o_M \} \). (ii) With agents’ state, they take actions in time step \( t \) based on their policy (i.e., actor), and then the state is transitioned to next time step \( t + 1 \) in MARL Layer. Here, all actions and rewards that agents take are denoted as the following sets, \( A \equiv \{ a_1, \cdots , a_m, \cdots , a_M \} \) and \( R \equiv \{ r_1, \cdots , r_m, \cdots , r_M \} \). (iii) Loss value is calculated by criticizing the reward that agents get at time step \( t \) with the return by target actor network in Trainer Layer. After that, the optimizer updates the parameters of actor networks in the direction of decreasing the value of the loss function. In the case of the target network, its parameters are updated at a specific time intermittently. The detailed training processes are described in Sec. III (iv) The Quantum layer, also known as the Q-policy network, produces the action distribution of all the agents by taking the state data as input. The output data is computed by the encoders and parameterized quantum circuits which are the components of the Q-policy network. Then, the action distribution values will be used to calculate the loss function utilized in the trainer layer via the mean squared error function. Finally, the gradient of the loss function is calculated to perform gradient descent to update the actor parameters and the target. (v) At the same time, all agent’s trajectories are visualized while training their policies by tensorboard in the visualization layer.

**Description of Q-policy.** Each agent has our proposed Q-policy as shown in Fig. 2. The algorithm for the Q-policy network is shown in Fig. 2(a). Firstly, (Lines 4–5) initialize the input dimensions as the size of the observation data of the agents and the initial quantum state dimension as the output size equivalent to the total number of actions. Next, (Line 6) defines the quantum device where the encoders and quantum layers will be placed. (Lines 7–9) and (Line 14) define the encoder and quantum layer, respectively. As shown in the figure, the encoder is composed of the pre-defined general encoder and re-uploading encoder, while the quantum layer is made up of Controlled U3 gates. In (Line 10), the number of repetitions of data re-uploading is set as the ceiling value of the input dimension divided by the number of initial qubits. (Lines 11–13) define the parameter values of the Controlled U3 gate. Finally, (Line 16) defines the measurement layer as applying Pauli-Z gate to all the qubits within the circuit. Fig. 2(a) contains the code for the forward function of the Q-policy network. (Lines 19–20) show the process of reshaping the input data into a single-column matrix. Then, (Line 21) is the part where data re-uploading occurs. The process is repeated for the number of repetitions initialized above. Every iteration passes the input data through one encoder and one quantum layer (Q-layer). Note that the first and the last iterations use encoders with different input dimensions. Lastly, (Lines 22–23) return the action distribution by measuring the output produced by data re-uploading. Fig. 2 also shows the structure of the Q-policy network, which is designed to perform data re-uploading. As shown in the figure, data re-uploading passes an initial quantum state through a repeated sequence of encoders and Q-layers. According to the algorithm elaborated above, the quantum gates in the encoders are imbued with the input state data. At the same time, the Q-layer is made up of Controlled U3 gates containing parameters such as the number of blocks, the number of layers per block, and the number of qubits. After data re-uploading has been performed, the output quantum state is measured by projecting the quantum state onto the reference z-axis to obtain the action distribution data. **Training Process of Agents.** Fig. 3 shows the training process of agents in our MARL environment with their policies. (Lines 2–9) perform the initialization of the policy’s parameters,
IV. PERFORMANCE EVALUATION

Visualization of Agents’ Training Results. Thanks to our visualization using tensorboard, we can analyze agents’ decisions over time. Fig. 4(a)–(c) show the total reward agents get, the support rate among all users, and users’ quality of service (QoS) in all epochs, respectively. All results show the common tendencies, where agents in the classical MARL get larger values in the intermediate step of learning and a faster increase rate in the early step of learning. However, users in the QMARL finally get more considerable value at the end of the training. In other words, our proposed QMARL shows a more stable convergence rate with little fluctuations and higher values at the end of the learning. We can also observe agents’ trajectories while learning their policies in Fig. 4(d)(f), where all agents try to provide high-quality wireless communication service to as many people as possible. When there is no malfunctioning, agents move to areas where any drone doesn’t provide service to users. Conversely, when any malfunction occurs, the agents move to areas where the malfunctioned non-agent has provided service to users. By our proposed visual simulation software framework, we can validate that the QMARL can make a reasonable performance with a smaller number of trainable parameters compared to classical MARL in reward convergence and system quality.

Demonstration Video. Note that the video demonstration for our proposed visual simulation software framework and simulation results are in [13].

V. CONCLUSION

We present a novel QMARL framework and analyze the performances by visualization using tensorboard. We show how our proposed QMARL could be used to train the multi-agent system efficiently. As future work, we plan to investigate other applications of QMARL in versatile situations.

Acknowledgement. This research was funded by National Research Foundation of Korea (2022R1A2C2004869). C. Park and J.P. Kim contributed equally to this work (first authors). W.J. Yun, S. Jung, and J. Kim are corresponding authors.

REFERENCES

[1] S. Jerbi, C. Gyurik, S. Marshall, H. Briegel, and V. Dunjko, “Parametrized quantum policies for reinforcement learning,” in Proc. NeurIPS, Virtual, December 2021.
[2] W. J. Yun, J. Park, and J. Kim, “Quantum multi-agent meta reinforcement learning,” in Proc. AAAI, Washington DC, USA, February 2023.
[3] W. J. Yun, Y. Kwak, J. P. Kim, H. Cho, S. Jung, J. Park, and J. Kim, “Quantum multi-agent reinforcement learning via variational quantum circuit design,” in Proc. IEEE ICDSC, Bologna, Italy, July 2022.
[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” arXiv preprint arXiv:1606.01540, 2016.
[5] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente et al., “Pettingzoo: Gym for multi-agent reinforcement learning,” in Proc. NeurIPS, vol. 34, Virtual, December 2021, pp. 15 032–15 043.
[6] W. J. Yun, S. Yi, and J. Kim, “Multi-agent deep reinforcement learning using attentive graph neural architectures for real-time strategy games,” in Proc. IEEE SMC, Virtual, October 2021.

Acknowledgement. This research was funded by National Research Foundation of Korea (2022R1A2C2004869). C. Park and J.P. Kim contributed equally to this work (first authors). W.J. Yun, S. Jung, and J. Kim are corresponding authors.

REFERENCES

[1] S. Jerbi, C. Gyurik, S. Marshall, H. Briegel, and V. Dunjko, “Parametrized quantum policies for reinforcement learning,” in Proc. NeurIPS, Virtual, December 2021.
[2] W. J. Yun, J. Park, and J. Kim, “Quantum multi-agent meta reinforcement learning,” in Proc. AAAI, Washington DC, USA, February 2023.
[3] W. J. Yun, Y. Kwak, J. P. Kim, H. Cho, S. Jung, J. Park, and J. Kim, “Quantum multi-agent reinforcement learning via variational quantum circuit design,” in Proc. IEEE ICDSC, Bologna, Italy, July 2022.
[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” arXiv preprint arXiv:1606.01540, 2016.
[5] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente et al., “Pettingzoo: Gym for multi-agent reinforcement learning,” in Proc. NeurIPS, vol. 34, Virtual, December 2021, pp. 15 032–15 043.
[6] W. J. Yun, S. Yi, and J. Kim, “Multi-agent deep reinforcement learning using attentive graph neural architectures for real-time strategy games,” in Proc. IEEE SMC, Virtual, October 2021.

[1] S. Jerbi, C. Gyurik, S. Marshall, H. Briegel, and V. Dunjko, “Parametrized quantum policies for reinforcement learning,” in Proc. NeurIPS, Virtual, December 2021.
[2] W. J. Yun, J. Park, and J. Kim, “Quantum multi-agent meta reinforcement learning,” in Proc. AAAI, Washington DC, USA, February 2023.
[3] W. J. Yun, Y. Kwak, J. P. Kim, H. Cho, S. Jung, J. Park, and J. Kim, “Quantum multi-agent reinforcement learning via variational quantum circuit design,” in Proc. IEEE ICDSC, Bologna, Italy, July 2022.
[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” arXiv preprint arXiv:1606.01540, 2016.
[5] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente et al., “Pettingzoo: Gym for multi-agent reinforcement learning,” in Proc. NeurIPS, vol. 34, Virtual, December 2021, pp. 15 032–15 043.
[6] W. J. Yun, S. Yi, and J. Kim, “Multi-agent deep reinforcement learning using attentive graph neural architectures for real-time strategy games,” in Proc. IEEE SMC, Virtual, October 2021.

[1] S. Jerbi, C. Gyurik, S. Marshall, H. Briegel, and V. Dunjko, “Parametrized quantum policies for reinforcement learning,” in Proc. NeurIPS, Virtual, December 2021.
[2] W. J. Yun, J. Park, and J. Kim, “Quantum multi-agent meta reinforcement learning,” in Proc. AAAI, Washington DC, USA, February 2023.
[3] W. J. Yun, Y. Kwak, J. P. Kim, H. Cho, S. Jung, J. Park, and J. Kim, “Quantum multi-agent reinforcement learning via variational quantum circuit design,” in Proc. IEEE ICDSC, Bologna, Italy, July 2022.
[4] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” arXiv preprint arXiv:1606.01540, 2016.
[5] J. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. S. Santos, C. Dieffendahl, C. Horsch, R. Perez-Vicente et al., “Pettingzoo: Gym for multi-agent reinforcement learning,” in Proc. NeurIPS, vol. 34, Virtual, December 2021, pp. 15 032–15 043.
[6] W. J. Yun, S. Yi, and J. Kim, “Multi-agent deep reinforcement learning using attentive graph neural architectures for real-time strategy games,” in Proc. IEEE SMC, Virtual, October 2021.

[1] S. Jerbi, C. Gyurik, S. Marshall, H. Briegel, and V. Dunjko, “Parametrized quantum policies for reinforcement learning,” in Proc. NeurIPS, Virtual, December 2021.
[2] W. J. Yun, J. Park, and J. Kim, “Quantum multi-agent meta reinforcement learning,” in Proc. AAAI, Washington DC, USA, February 2023.
[13] C. Park, J. P. Kim, W. J. Yun, S. Jung, and J. Kim. (2022, November) Visual simulation software demonstration for quantum multi-drone reinforcement learning. [Online]. Available: https://sites.google.com/view/icse-2023