A Reinforcement Learning-Based Framework for Multi-Agent Path Planning and Information Sharing

Anonymous CVPR submission

Paper ID *****

Abstract

Multi-agent systems have gained significant attention in robotics, computer vision, and natural language processing due to their potential to enhance task efficiency, robustness, and scalability. Among the challenges in multi-agent systems, path planning plays a pivotal role as it determines agents' trajectories and interactions, impacting overall system performance. This paper presents a reinforcement learning-based strategy for multi-agent path planning, emphasizing information sharing and collaboration. By designing a shareable "common language," we enable effective communication and decision-making among agents, improving exploration and path planning efficiency in unknown environments.

Specifically, we propose a framework that incorporates shared Q-matrices, distributed region-based information integration, and convolutional autoencoders for data compression. Experimental results demonstrate that shared Q-matrices significantly enhance task completion efficiency across different starting points and validate the feasibility of information sharing across separate regions. While the performance of convolutional autoencoders in high-dimensional data compression reveals room for improvement, future work will explore vision transformer-based approaches to further optimize path planning and system performance.

Our dataset and code of the experiment is available at `https://github.com/NZJ-Jimmy/RL-Multi-Agent-Maze`.

1. Introduction

Multi-agent systems have been widely studied in various fields, such as robotics, computer vision, and natural language processing, due to their potential to improve task efficiency, robustness, and scalability. In multi-agent systems, agents need to communicate and collaborate to achieve common goals, which requires effective information sharing and coordination strategies. Path planning is a fundamental problem in multi-agent systems, as it determines the agents' trajectories and interactions, affecting the overall system performance. In this paper, we focus on path planning strategies for multi-agent information sharing and collaboration based on reinforcement learning.

Information communication is essential for multi-agent systems to share knowledge and coordinate actions. In the "Teacher-to-student communication model" [5], the teacher agent compresses complex task information into low-dimensional messages using a sparse autoencoder, forming a common language. The student agents then learn to decode these messages to improve task completion rates and generalization capabilities. This shared language-based communication mechanism not only facilitates effective information transmission, but also optimizes the content of the language through feedback from student agents, enhancing collaboration and learning abilities among agents. This simulates the evolution and function of natural language in social interactions.

Collaboration is another critical aspect of multi-agent systems, where agents need to explore the environment together to discover and achieve common goals. Large language models (LLMs) have shown promising performance in multi-agent cooperative tasks, especially in reasoning abilities involving "Theory of Mind (ToM)" [4]. LLMs are evaluated in a text-based game environment simulating a search and rescue mission, where agents collaborate to locate and defuse bombs.

In this paper, multiple agents explore different parts of a map based on reinforcement learning and need to exchange the information they have obtained to complete the task of exploring the entire map.

We are going to design a shareable "common language" so that the information obtained by a single agent can be "translated" into this language to provide guidance for the decision-making of other agents. Using this "common language" for information sharing and collaboration among multiple agents completes the task of path planning in the map.

To summarize, we present a framework for multi-robot exploration in unknown environments, utilizing abstract message data and shared information variables to achieve more efficient path planning. Our contributions to the field of multi-agent systems are as follows:

- Develop a method for multiple agents to share and merge information, creating a unified representation of the environment and designing a “common language” that enables effective exchange of information from maps between agents.
- Enhance the multi-robot system’s adaptability and accuracy in path planning by leveraging shared data, which allows for quicker and more effective path planning in complex, unknown spaces.

2. Related Work

2.1. Teacher-to-student communication model

Teachers and students communicate through a shared language framework: teachers encode complex task knowledge (such as Q-matrix) into low-dimensional messages through reinforcement learning, and students learn to decode this information to make better decisions and execute tasks to improve task completion rates and generalization capabilities. This communication mechanism enhances the efficiency of information transmission, promotes collaboration and learning among agents, and highlights the importance of a common language in task generalization and information transmission. [5]

This mechanism reflects the compression and transmission of information, achieving low-dimensional representation of complex information through sparse autoencoders, ensuring that the information retains key task relevance. Student feedback is used to optimize the message content, ensuring the practicality and adaptability of the common language. Although the description focuses on the unidirectional transmission from teacher to student, it also implies the potential for bidirectional communication, where student feedback can influence the teacher’s language generation, forming a dynamic learning environment.

2.2. Multi-Agent Collaboration via LLMs

This study [4] aims to explore the performance of large language models (LLMs) in multi-agent cooperative tasks, especially in the reasoning ability involving “Theory of Mind (ToM)”. The research background points out that although LLMs have made remarkable achievements in reasoning and planning, their ability in multi-agent collaboration has not been fully studied.

The ability of LLMs in multi-agent cooperative tasks is evaluated in a text-based game environment. The game simulates a search and rescue mission, requiring agents to collaborate to locate and defuse bombs, which requires agents to demonstrate the ability of understanding the environment, task planning, action execution, and feedback interpretation. LLMs are used as agents and compared with multi-agent reinforcement learning (MARL) and planning-based methods. The results show that LLMs show signs of cooperative behavior and high-order ToM ability, but also have limitations, especially in managing long-term context and avoiding hallucinations about task status.

3. Method

3.1. Q-Learning Algorithm

Q-Learning [3] is a value-based reinforcement learning algorithm, and its core is to update the value of state action pairs (Q values) so that the agent can learn the optimal policy in the environment. The core update formula of Q-Learning is as follows:

$$Q(s_t, a) := Q(s_t, a_t) + \alpha \cdot [R(s_t, a) + \gamma \cdot \max Q(s_{t+1}, a') - Q(s_t, a)] \tag{1}$$

In Eq. (1), the meanings of the various variables and parameters are as follows:

- $Q(s_t, a)$ is the Q-value of state s_t and action a at the time step t.
- α is the learning rate, which controls the trade-off between the new estimated value and the old estimated value, with a range of $0 \leq \alpha \leq 1$. The learning rate determines the degree of influence of new information on the update of the Q value.
- $R(s_t, a)$ is the immediate reward obtained after executing the action$a$.
- γ is the discount factor, indicating the importance of future rewards, with a range of $0 \leq \gamma \leq 1$. The discount factor reflects the importance of future rewards relative to immediate rewards.
- s_{t+1} is the new state observed after executing the action a.
- $\max Q(s_{t+1}, a')$ is the maximum Q-value of all possible actions in the new state s_{t+1}, representing the maximum expected future reward.

3.2. Randomized Prim’ s Maze Generation Algorithm

The Randomized Prim’ s maze generation algorithm [1] is a method for generating mazes based on Prim’ s minimum-spatial tree algorithm. The core idea is to represent the maze as a graph, where each cell (or ”room”) is a node, and each node is connected to its adjacent nodes via edges. By randomly assigning weights to the edges and generating a minimum spanning tree, the algorithm forms the paths of the maze. This method ensures the generation of a loop-free maze, guaranteeing that there is a unique shortest path from any point to any other point.

The Fig. 1 shows the initial maze used by the algorithm. The outermost boundary of the maze is by default a ring of walls. White cells represent maze cells, and gray cells represent walls. The walls between adjacent white cells can be removed. It can be seen that the positions (X, Y) of all maze cells in the maze are odd numbers, such as (1, 3) and (5, 7), which can be expressed as $(2x+1, 2y+1)$, where the range of values for x and y is from 0 to 3. This representation is used in the maze generation algorithm. Additionally, the length and width of the maze must be odd numbers.

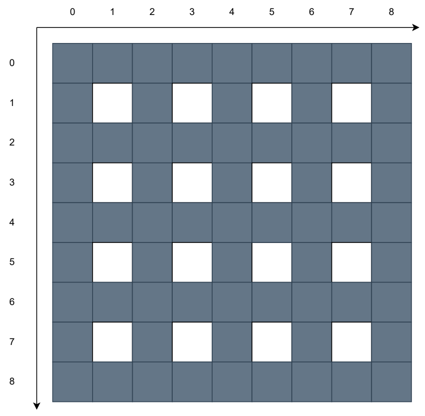


Figure 1. The initial maze

The algorithm main loop is as follow, repeat the Step 2 until the check list is empty:

- Step 1: Randomly select a maze cell as the starting point, add it to the check list, and mark it as visited.
- Step 2: While the check list is not empty, randomly remove a maze cell from the list and perform the loop.
  - If the current maze cell has unvisited adjacent maze cells:
    - ∗ Randomly select an unvisited adjacent maze cell.
    - ∗ Remove the wall between the current maze cell and the adjacent maze cell.
    - ∗ Mark the adjacent maze cell as visited and add it to the check list.
  - Otherwise, if the current maze cell has no unvisited adjacent maze cells:
    - ∗ Remove the current maze cell from the check list.

3.3. Distance Matrix

In multi-agent path planning, each agent starts from a unique starting point, and the number of steps required to reach other accessible locations within the maze varies.To efficiently calculate these steps, we use the Breadth-First Search (BFS) algorithm to determine the minimum number of steps each agent needs to take from its starting point to all other accessible locations within the maze. Subsequently,these step counts are filled into a distance matrix, where the entries corresponding to accessible locations within the maze are the minimum number of steps required by any of the multiple agents to reach that location. The specific steps are as follows:

- **Initialize the local distance matrix for each agent**
  - For each agent, initialize a distance matrix $\boldsymbol{D}_i$, where $\boldsymbol{D}_i(s, g)$ represents the minimum number of steps agent i needs to take from the starting point s to reach location g within the maze. Initially, set all distances to walls as infinity (∞), indicating inaccessibility, and set distances to other accessible locations as -1, indicating accessibility but unknown step count. Finally, set the distance from the starting point to itself as 0.
- **Breadth-First Search (BFS)**
  - For each agent i, perform the BFS algorithm starting from its starting point s_i.
  - Use a queue to store nodes to be processed, initially containing only the starting point s_i.
  - While the queue is not empty, perform the following steps:
    - ∗ Remove a node u from the queue.
    - ∗ For each neighbor node v of node u, if v has not been visited:
      - · Calculate the distance from the starting point s_i to node v as $\boldsymbol{D}_i(s_i, v) = \boldsymbol{D}_i(s_i, u) + 1$.
      - · Mark node v as visited and add it to the queue.
- **Update the global distance matrix**
  - Initialize a global distance matrix $\boldsymbol{D}$, where $\boldsymbol{D}(g)$ represents the minimum number of steps required by all agents to reach location g within the maze. Initially, set all distances to infinity (∞).
  - For each accessible location g within the maze, perform the following steps:
    - ∗ Traverse the distance matrices $\boldsymbol{D}_i$ of all agents to find the minimum number of steps to reach location g, $d_{\text{min}} = \min_i(\boldsymbol{D}_i(s_i, g))$.
    - ∗ Fill d_{min} into the global distance matrix $\boldsymbol{D}(g)$.

Through the above steps, we can obtain a global distance matrix $\boldsymbol{D}$, where the value at each position g represents the minimum of the minimum steps required for each agent to reach that position.

3.4. Convolutional Autoencoder

Taking a maze with dimensions (9, 9) and considering a scenario with 2 agents as an example, the Convolutional Autoencoder designed in this paper is used to encode and compress two Q-matrices (each with dimensions $(4, 9, 9)$) obtained by two robots exploring and navigating to the same destination within a maze using Q-learning algorithm, into a low-dimensional representation m (where m is a real-valued vector of length K). This low-dimensional representation is then decoded to generate a single Q-combined-matrix (with dimensions $(4, 9, 9)$). It is hoped that this approach will enable more efficient path representation and optimization in multi-agent path planning. The input dimension $x_{\dim}$ is (2×4, 9, 9), representing the two Q-matrices, and the output dimension $y_{\dim}$ is (4, 9, 9), representing the single Q-combined-matrix. See the model architecture in Fig. 2.

Convolutional Autoencoder can be divided into three main components as follow:

1. **Input Layer**
   - The input dimension is $(8, 9, 9)$, representing two Q-matrices (each Q-matrix has dimensions$(4, 9, 9)$).
   - enc_bias: BiasLayer, used to add a bias term.
2. **Encoder**
   - enc_conv_1: The first convolutional layer, mapping the input from $(8, 9, 9)$ to $(10, 10, 10)$.
   - enc_conv_2: The second convolutional layer, mapping the input from $(10, 10, 10)$ to $(10, 11, 11)$.
   - enc_flatten:Flattening the output of the convolutional layers from $(10, 11, 11)$ to (1210).
   - enc_lin:A fully connected layer, mapping the flattened output from (1210) to (30).
3. **Decoder**
   - dec_lin: A fully connected layer, mapping the output of the encoder from (30) back to (1210).
   - dec_unflatten: Reshaping the output of the fully connected layer from (1210) to $(10, 11, 11)$.
   - dec_deconv_1: The first transposed convolutional layer, mapping the input from $(10, 11, 11)$ to $(10, 10, 10)$.
   - dec_deconv_2: The second transposed convolutional layer, mapping the input from $(10, 10, 10)$ to $(4, 9, 9)$.
   - dec_bias: BiasLayer, used to add a bias term.

4. Experiment

To evaluate the sharing performance of the algorithms, we conduct experiments on abstract mazes with agents.

- The maze is a $n \times m$ grid with walls and a destination.
- The agent can move in four directions: up, down, left, and right.
- The agent receives a reward of -1 for each step, and a reward of 100 for reaching the destination.
- The agent receives a reward of -100 for hitting the wall. The agent will stay in the same position if it hits the wall.

4.1. Q-Matrix Shared with Multi-Agent

A well-trained Q-Matrix can guide an agent from any points to a specific end point, by following the path with the highest Q-Value. However, the Q-Matrix is trained by a single agent with a specific starting point and end point. It may only effectively guide the agent to reach the destination from the same starting point. If the agent starts from a different point, the Q-Matrix may not be able to instruct the agent to reach the destination. In this case, the agent needs to explore the environment to find the path to the destination.

To solve this problem, we assume that the Q-Matrix can be shared among multiple agents. We propose a method to share the Q-Matrix among multiple agents while training. The shared Q-Matrix is trained by each agent with a different starting point to the same end point, before agents use it to guide them to reach the destination from any starting points.

4.1.1. Sample Settings

We use a simple maze 10×10 to demonstrate the effectiveness of the shared Q-Matrix. The maze is shown in Fig. 3, with a end point at $E(4, 5)$ and walls around the maze. See Fig. 3.

4.1.2. Training

To compare the performance of the shared Q-Matrix and the individual Q-Matrix, we train the agents with the following settings:

1. **Individual Q-Matrix**: A single agent is trained with the starting point at $S_1(8, 1)$ and the end point at $E(4, 5)$.
2. **Shared Q-Matrix**: Multiple agents are trained with 2 starting points in order: $S_1(8, 1)$ and $S_2(8, 8)$, and the same end point at $E(4, 5)$(see Fig. 3). They share the Q-Matrix during training.

For each setting, we train the agents with the same hyperparameters:

- The agent is trained with 500 epochs.
- The agent uses the greedy policy with $\epsilon = 0.3$.
- The agent uses the Q-Learning algorithm with $\alpha = 0.1$ and $\gamma = 0.9$ in Eq. (1).

4.1.3. Test

After training, we test the agents with 2 starting points $S_1(8, 1)$ and $S_2(8, 8)$, setting the greedy policy with $\epsilon = 0$. We compare the performance of the agents with the individual Q-Matrix and the shared Q-Matrix, to evaluate the effectiveness of the shared Q-Matrix.

1. **Starts from** $S_1(8, 1)$: The agent starts from $S_1(8, 1)$ and uses the two Q-Matrices to reach the destination $E(4, 5)$.
   - **Individual Q-Matrix**: Got the path: $S_1(8, 1) \rightarrow (8, 2) \rightarrow (8, 3) \rightarrow (7, 3) \rightarrow (6, 3) \rightarrow (6, 4) \rightarrow (6, 5) \rightarrow (5, 5) \rightarrow E(4, 5)$. *The agent reaches the destination.*
   - **Shared Q-Matrix**: Got the path: $S_1(8, 1) \rightarrow (7, 1) \rightarrow (6, 1) \rightarrow (6, 2) \rightarrow (6, 3) \rightarrow (6, 4) \rightarrow (6, 5) \rightarrow (5, 5) \rightarrow E(4, 5)$. *The agent reaches the destination.*
2. **Starts from** $S_2(8, 8)$: The agent starts from $S_2(8, 8)$ and uses the two Q-Matrices to reach the destination $E(4, 5)$.

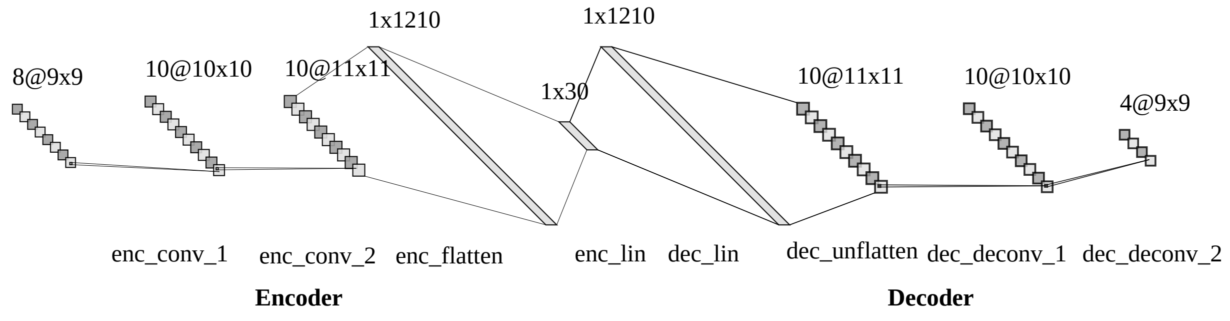


Figure 2. The architecture of Convolutional Autoencoder

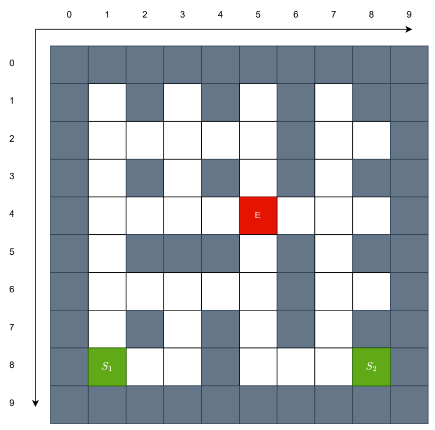
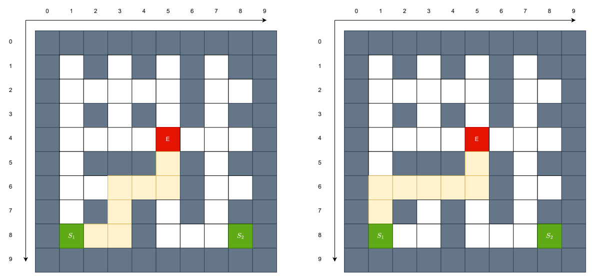


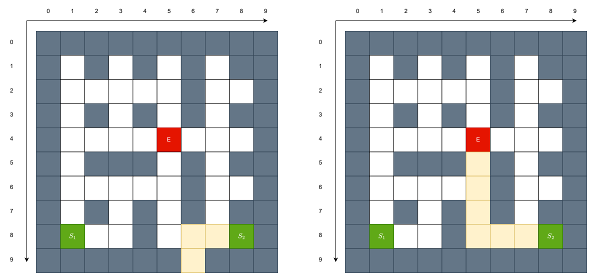
Figure 3. A sample maze for the multi-agent Q-Matrix sharing. *The agent needs to reach the red grid. The gray grids are walls.*



(a) Path of Individual Q-Matrix. (b) Path of Shared Q-Matrix.

Figure 4. The path of the agent starts from $S_1(8, 1)$. *The yellow grids are the path of the agent.*

- **Individual Q-Matrix**: Got the path: $S_2(8, 8) \to (8, 7) \to (8, 6) \to (9, 6)$. *The agent hits the wall.*
- **Shared Q-Matrix**: Got the path: $S_2(8, 8) \to (8, 7) \to (8, 6) \to (8, 5) \to (7, 5) \to (6, 5) \to (5, 5) \to E(4, 5)$. *The agent reaches the destination.*



(a) Path of Individual Q-Matrix (b) Path of Shared Q-Matrix.

Figure 5. The path of the agent starts from $S_2(8, 8)$. The yellow grids are the path of the agent. *The yellow grids are the path of the agent.*

4.1.4. Analysis

From the test results, we can see that the shared Q-Matrix can guide the agent to reach the destination from different starting points, while the individual Q-Matrix may not be able to guide the agent to reach the destination from different starting points, especially when the starting point is far from the training path. The shared Q-Matrix can be used to guide the agent to reach the destination from more starting point, which is more flexible and effective than the individual Q-Matrix.

4.2. Q-Matrix Shared in Different Area

As we have shown that the shared Q-Matrix can guide the agent to reach the destination from different starting points, we confirm that the Q-Matrix can be shared among agents with different starting points.

However, the shared Q-Matrix is trained by agents with starting points in the same area. In reality, the area that each agent can explore may be restricted, and they may not all explore the same area.

We wonder if the Q-Matrix can be shared among agents with in different areas that they explore.

4.2.1. Sample Settings

We use the same maze settings in Sec. 4.1, split the maze into 4 areas, and train the agents with different starting points at the corner, shown in Fig. 6. The agents can only explore the area that they start from, targeting the same end point at $E(4, 5)$, and they share the Q-Matrix during training.

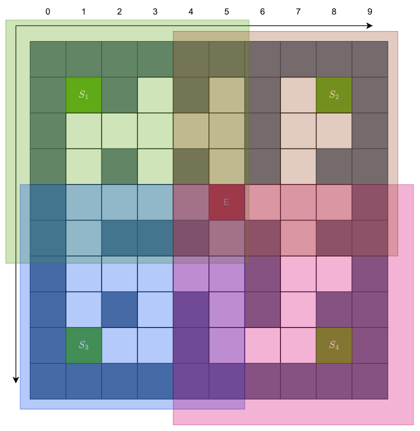


Figure 6. The maze is split into 4 areas. *The agents(green) start from the corners, can only walk in the area in same color, targeting the red grid.*

4.2.2. Training

Four agents are trained with the following settings:

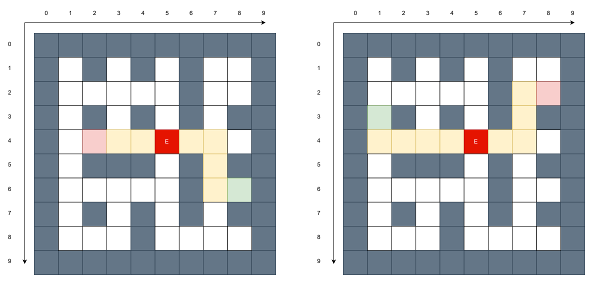
- The agents are trained with 100 epochs.
- The agents use the greedy policy with $\epsilon = 0.3$.
- The agents use the Q-Learning algorithm with $\alpha = 0.1$ and $\gamma = 0.9$ in Eq. (1).
- The agents is restricted to explore the area that they start from. It assume that the grids out of its area are walls.

4.2.3. Test

Assume a scenario: an agent needs to carry something from the starting point S_i to the passing point E, and then another agent takes it back from the passing point E to S_j. The path can be calculated by the shared Q-Matrix by following steps:

1. The first agent uses the shared Q-Matrix to get the path $S_i \to E$.
2. The second agent uses the shared Q-Matrix to get the path $S_j \to E$.
3. Reverse the path of the second agent, and combine it with the path of the first agent. The path is $S_i \to E \to S_j$.

By planning the path crossing the areas, we can verify the effectiveness of the shared Q-Matrix among agents in different areas.



(a) Path of the agent from $(6, 8)$ to $(4, 1)$. (b) Path of the agent from $(3, 1)$ to $(2, 8)$.

Figure 7. The path of the agents in different areas. *The yellow grids are the path of the agent.*

Here are the test results, the paths are shown in Fig. 7.

- **Planning the path from** $(6, 8)$ **to** $(4, 1)$: Got the path: $(6, 8) \to (6, 7) \to (5, 7) \to (4, 7) \to (4, 6) \to (4, 5) \to (4, 4) \to (4, 3) \to (4, 2) \to (4, 1)$.
- **Planning the path from** $(3, 1)$ **to** $(2, 8)$: Got the path: $(3, 1) \to (4, 1) \to (4, 2) \to (4, 3) \to (4, 4) \to (4, 5) \to (4, 6) \to (4, 7) \to (3, 7) \to (2, 7) \to (2, 8)$

4.2.4. Analysis

From the test results, we can see that the shared Q-Matrix can guide the agents to reach the destination from different areas, which means that the Q-Matrix can be shared among agents across different areas.

4.3. Combine Q-Matrices with Convolutional Autoencoder

The Q-Matrix is a table that stores the Q-Values of each state-action pair. It is a high-dimensional table that requires a lot of memory to store. And each agent needs to store a Q-Matrix for training. As we have shown that one Q-Matrix can be shared among agents across different areas, we wonder if Q-Matricies trained by agents can be compressed by a Convolutional Autoencoder, and then shared among agents.

4.3.1. Dataset

To train the Convolutional Autoencoder, we generate data from the Q-Matrix trained by agents. The data is a set of individual Q-Matrices and its shared Q-Matrices as in Sec. 4.1, from specific starting points to the each end point.

1. **Generate mazes**: Generate 100 mazes with the same size 9×9. The mazes are generated as Sec. 3.2.
2. **For mazes**: Calculate its distance matrix as in Sec. 3.3. Traverse every reachable point as the end point.
3. **For end points**: Train agents as in Sec. 4.1.2.
   - Train 2 agents with the specific starting points individually, and get the individual $\boldsymbol{Q}_i$ of i-th agents.
   - Train 2 agents with the specific starting points together in a shared Q-Matrix, and get the shared $\boldsymbol{Q}_{\text{combo}}$.

Finally, we got 100 batches of data, each batch is generated from a maze. Each batch contains 2 individual Q-Matrices and 1 shared Q-Matrix, packaged as two Tensors `q_matrices_tensor` and `q_matrix_combo_tensor` with sizes of $(100, 4{\times}2, 9, 9)$ and $(100, 4, 9, 9)$ respectively.

4.3.2. Training

We split the dataset into training and testing sets with a ratio of 8:2, and train the Convolutional Autoencoder with the following settings:

- Train with 100 epochs.
- Set $K = 30$ as the message length/hidden dimension.
- Use the Mean Squared Error(MSE) loss, comparing the reconstructed Q-Matrices $\boldsymbol{Q}_{\text{rec}}$ with the shared Q-Matrices $\boldsymbol{Q}_{\text{combo}}$.
- Use the Adam optimizer with a learning rate of 0.001.

Although the loss is gradually decreasing during training and the model seems to be gradually fitting, the loss is still high (42.20). See Fig. 8.

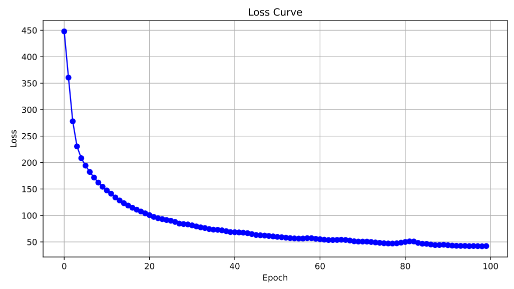


Figure 8. The loss of the Convolutional Autoencoder during training.

4.3.3. Test

We use the trained Convolutional Autoencoder to compress $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ of the agents and get the compressed Q-Matrices $\boldsymbol{Q}_{\text{rec}}$. And we compare the $\boldsymbol{Q}_{\text{rec}}$ with the shared Q-Matrix $\boldsymbol{Q}_{\text{combo}}$ using the MSE loss. See Fig. 9.

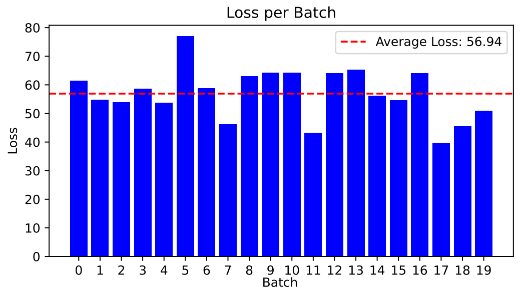


Figure 9. The loss of the Convolutional Autoencoder on the testing set.

The loss is 56.94 in average, which is about 1.3 times the loss of the training set. The result shows that the Convolutional Autoencoder can compress the Q-Matrix with a certain loss. However, the loss is still high. We wonder if the $\boldsymbol{Q}_{\text{rec}}$ can be used to guide the agent to reach the destination.

We test the agents with the compressed Q-Matrices $\boldsymbol{Q}_{\text{rec}}$, starts from the original starting points to the end points, and get its path length. And uses the distance matrix to get the shortest path length, comparing the path length of the agents with $\boldsymbol{Q}_{\text{rec}}$. See the results in Fig. 10.

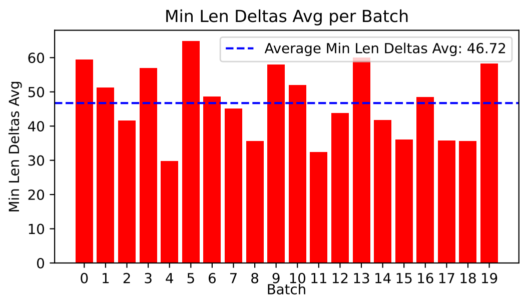


Figure 10. The path length of the agents with $\boldsymbol{Q}_{\text{rec}}$ minus the shortest path length.

The result shows that the path length of the agents with $\boldsymbol{Q}_{\text{rec}}$ is about 50 steps further than the shortest path, which means that the agents cannot reach the destination with the compressed Q-Matrix.

5. Discussion

For the results of several experiments, we have the following discussion:

- Q-Learning is a powerful algorithm that can learn an optimal strategy through continuous trials without any prior knowledge. In our experiments, Q-Learning performed well in various environments, demonstrating the ability to learn information across agents and areas. The learned information can be shared among multiple agents for training and testing, achieving a certain degree of multi-agent collaboration to find the optimal strategy in complex environments.
- The performance of the Convolutional Autoencoder did not meet our expectations. In our experiments, we found that the Convolutional Autoencoder struggled to fit the original information during training, resulting in poor model performance. This could be due to the high number of parameters in the Convolutional Autoencoder, making it difficult to train, or the low dimensionality of the encoded information, limiting the decoding capability. In future work, we can try using more techniques to improve the performance of the Convolutional Autoencoder,

such as adjusting the model's hyper-parameters, or directly discarding the decoder and convolving the individual Q-Matrix into the reconstructed Q-Matrix.

- Considering the characteristics of the maze, we thought that the Vision Transformer (ViT)[2] model might be good at extracting information from images. In future work, we will try using the ViT model, inputting the Q-Matrix learned by the agent, as well as the start and end points, to let the ViT model learn the structure of the maze and directly output the optimal path.

6. Conclusion

In this study, we proposed a reinforcement learning-based framework for multi-agent path planning and information sharing, addressing critical challenges in multi-agent systems such as communication, collaboration, and exploration in unknown environments. Our approach facilitates the efficient exchange and integration of information, supported by a shared Q-matrix mechanism to enhance task performance across multiple agents and regions.

Experimental results demonstrated that shared Q-matrices significantly improve the adaptability and effectiveness of multi-agent systems, enabling agents to navigate and collaborate efficiently in complex, distributed environments. However, the application of convolutional autoencoders for compressing Q-matrices revealed certain limitations, highlighting the need for further optimization.

Future work will focus on advancing information encoding and sharing strategies, including the use of Vision Transformers (ViT) to better capture and process spatial relationships in path planning tasks.

References

[1] V. Bellot, M. Cautrès, J-M. Favreau, M. Gonzalez-Thauvin, P. Lafourcade, K. Le Cornec, B. Mosnier, and S. Rivière-Wekstein. How to generate perfect mazes? *Information Sciences*, 572:444–459, 2021. 2

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 8

[3] Beakcheol Jang, Myeonghwi Kim, Gaspard Harerimana, and Jong Wook Kim. Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, 7:133653–133667, 2019. 2

[4] Huao Li, Yu Chong, Simon Stepputtis, Joseph Campbell, Dana Hughes, Charles Lewis, and Katia Sycara. Theory of Mind for Multi-Agent Collaboration via Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 180–192, Singapore, 2023. Association for Computational Linguistics. 1, 2

[5] Tobias J. Wieczorek, Tatjana Tchumatchenko, Carlos Wert-Carvajal, and Maximilian F. Eggl. A framework for the emergence and analysis of language in social learning agents. *Nature Communications*, 15(1):7590, 2024. 1, 2