



Development of agent-based mesh generator for flow analysis using deep reinforcement learning

Keunoh Lim¹ · Kyungjae Lee² · Sanga Lee³ · Kwanjung Yee¹

Received: 28 May 2024 / Accepted: 31 July 2024 / Published online: 11 August 2024
© The Author(s) 2024

Abstract

Computational fluid dynamics (CFD) has widespread application in research and industry. The quality of the mesh, particularly in the boundary layer, significantly influences the CFD accuracy. Despite its importance, the mesh generation process remains manual and time intensive, with the introduction of potential errors and inconsistencies. The limitations of traditional methods have prompted the recent exploration of deep reinforcement learning (DRL) for mesh generation. Although some studies have demonstrated the applicability of DRL in mesh generation, they have limitations in utilizing existing tools, thereby falling short of fully leveraging the potential of DRL. This study proposes a new boundary mesh generation method using DRL, namely an agent-based mesh generator. The nodes on the surface act as agents and optimize the paths into space to create high-quality meshes. Mesh generation is naturally suited to DRL owing to its computational nature and deterministic execution. However, challenges also arise, including training numerous agents simultaneously and managing their interdependencies in a vast state space. In this study, these challenges are addressed along with an investigation of the optimal learning conditions after formulating grid generation as a DRL task: defining states, agents, actions, and rewards. The derived optimal conditions are applied to generate two dimensional airfoil grids to validate the feasibility of the proposed approach.

Keywords Mesh generation · Deep reinforcement learning · Computational fluid dynamics · Deep Q learning

1 Introduction

Computational fluid dynamics (CFD) has been applied to a wide range of research and industrial problems, such as aerodynamic simulation [1, 2], aerospace analysis [3, 4], and gas dynamics prediction [5, 6]. Although the entire CFD analysis process has now been largely automated, human involvement remains necessary in the mesh generation part. The mesh generation process, which involves subjective decision-making and manual intervention, requires considerable time and effort, with potential errors and inconsistencies introduced into the CFD analysis [7].

The quality of the mesh on the boundary layer is particularly important because it plays a crucial role in accurately capturing the intricacies of viscous flow, and thus, directly impacts the fidelity and precision of subsequent flow analyses [8, 9]. In particular, the generation of meshes for the boundary layer with intricate geometries poses significant challenges. These geometries require time-consuming techniques to generate suitable meshes. Therefore, developing efficient and robust methods for mesh generation in various surface geometries is paramount for removing researcher bias from the accuracy of CFD results and reducing the working time in various engineering applications.

Several studies have been conducted to overcome the challenges associated with mesh generation. Marshall et al. [10] reported on methods that are widely used for structured grid generation in open-domain simulations, such as fluid dynamics. In this study, structured grid generation methods were classified into algebraic, interpolation, PDE[A2], and variational methods. Representative PDE methods include the hyperbolic and elliptic methods. The research of Sørensen [11] is a prominent example of hyperbolic

Keunoh Lim and Kyungjae Lee have contributed equally to this work.

This work was presented at the AIAA SciTech 2024 Forum, 8–12 January 2024, Orlando, Florida (Lim, Lee, Lee, and Yee 2024). While the key ideas were initially presented at the forum, significant improvements and enhancements have been made since then.

Extended author information available on the last page of the article

methods. In this study, hyperbolic mesh generation methods were reported to be advantageous for rapidly and robustly creating highly orthogonal grids. However, limitations in the geometry that can be applied were noted. In terms of elliptic methods, Thompson et al. [12] reported that elliptic PDEs are generally more suitable for complex geometries than hyperbolic methods. However, Thomas et al. [13] and Steger et al. [14] pointed out that elliptic mesh methods require precise parameter selection.

These methods have limitations in addressing the problems associated with grid generation because of the significant human intervention required to create sophisticated grids. Some researchers have recently shown interest in using deep reinforcement learning (DRL) to solve mesh generation problems. DRL is a promising machine learning approach that enables an intelligent agent to optimize the sequential decision-making process through trial and error [15]. In DRL, an agent explores a set of possible decisions and receives rewards based on an evaluation of the decisions made by the agent. The goal of a DRL agent is to determine the optimal sequence of decisions that maximizes the cumulative rewards; that is, to identify the optimal path [16]. This method has achieved considerable success in various decision-making problems, such as path planning [15, 17], autonomous driving [18, 19], and robotics [20], by demonstrating its capability to manage complex decision-making scenarios.

Zhang et al. [21] and Pan et al. [22] conducted research on generating grids in closed domains for the finite element analysis and leveraging the strengths of DRL. Kim et al. [23] utilized DRL for grid generation in fluid analysis. In this study, DRL-based multi-condition optimization techniques were used and optimal mesh parameters were efficiently derived when the blade geometry was input. These studies demonstrated the successful application of DRL to grid generation problems. However, these methods have limitations in fully utilizing the advantages of DRL. The strengths of DRL lie in the continuous decision-making and optimization of processes. However, these studies used DRL to select the final parameters rather than continuous decision-making. Furthermore, because these studies optimized the parameters of existing mesh generation tools, they cannot be confirmed to have outperformed traditional mesh generation methods.

Considering these factors, our study proposes a completely new boundary mesh generation method using DRL. The fundamental concept is to redefine the mesh generation problem as an optimization problem in which the nodes on the surface extend into space and optimize their paths. Given information regarding the surface shape, each node acts as an agent, learning to move towards the outer space such that the path that it creates results in a high-quality mesh.

The mesh generation problem has several features that make it highly suitable for redefinition using DRL. First,

to apply DRL, a separate simulation system must be created for learning. However, because mesh generation is inherently a computational system, a separate environment does not need to be configured. Second, there is no uncertainty in the action execution of the agent. When training DRL models for autonomous driving or robotics, the possibility that the actions taken by the agent may not be properly executed owing to unforeseen circumstances must be considered. However, these issues do not need to be considered in grid problems. These characteristics provide significant advantages for addressing mesh generation problems using DRL.

However, several issues need to be addressed. First, the reward is not determined by the actions of a single agent. In classical DRL problems, agents receive rewards for their actions. However, during grid generation, the action of a single agent does not correspond to a single reward because the trajectory created by the movement of a single node does not result in the generation of a grid. The second issue is the large dimensionality of both the agent and state. As mentioned previously, the nodes on the surface function as both agents and states that serve to move themselves and provide information for decision-making. Typically, many surface nodes are employed for grid generation, implying that a vast number of agents must be trained using an extensive set of state information. This poses a significant challenge compared with dealing with only one or several agents in autonomous driving or robotics problems. The third issue is the high level of dependency among agents. The paths created by multiple nodes must be optimized simultaneously to create an effective grid.

In this study, we first formulate the grid generation problem as a DRL task to implement the proposed concept. This involves transforming the grid generation problem into a DRL framework that consists of defining the state, agent, action, and reward. Next, we report various efforts to ensure that the DRL is properly trained, including addressing the issues mentioned previously. This includes research on modifying the training process and introducing sub-episodes within the main episode loop to handle non-deterministic rewards, as well as defining the state representation to address the issue of dimensionality of the agent and state. In addition, penalty policies and various learning strategies are explored with the aim of mitigating the interdependence of actions. Based on these investigations, optimal learning conditions are proposed and applied to generate airfoil grids. Thus, the concept of a new agent-based mesh generator is validated.

Due to the challenging nature of finding suitable learning conditions in DRL, which requires numerous case studies on various learning parameters and methods, a DRL framework was established based on a two dimensional example with fewer agents, i.e., nodes, to effectively develop the learning

strategy. Instead, Sect. 4.7 of this paper will briefly mention the extension to three dimensions.

2 Methodology

2.1 Reinforcement learning

In reinforcement learning (RL), the interaction between an agent and the environment is a fundamental concept [24] that defines how the agent acquires knowledge and adjusts its behavior based on the feedback provided by the environment. As shown in Fig. 1, the interaction typically occurs in an iterative manner, involving the following components:

The agent is the learner or decision-maker in the RL process. The environment is the external system with which the agent interacts. Thus, the learning process consists of continuous interactions between the agent and environment. For each time step t , the environment provides the state S_t to the agent. The state S_t represents the current situation or configuration of the environment at time step t . It contains all information required for the agent to make decisions. Based on S_t , the agent selects A_t , which affects the state of the environment and leads to a transition from one state to another. The agent follows a specific action selection rule known as a policy π , which is a mapping from states to actions; that is, $\pi(S_t) = A_t$. The behavior of the agent is fully defined by π . Subsequently, as a result of the decision made by the agent, the environment returns a reward R_{t+1} and the next state S_{t+1} . The reward R_{t+1} is a numerical signal provided by the environment to the agent following each action. It quantifies the desirability or quality of the agent action in a specific state. By repeating the same procedure for S_{t+1} , the agent accumulates information regarding the transitions of the environment and the quality of the instant decision.

The goal of the agent is to maximize the cumulative rewards acquired over the time steps. As the agent in RL is generally assumed to have no prior knowledge of the dynamics of the states and rewards, the agent must explore various states and actions to discover the optimal sequence of actions. This exploration is a critical challenge in RL as it involves the agent learning from trial and error without

explicit guidance on the best actions to take. To address this challenge, several strategies have been employed to balance the trade-off between exploration (attempting new actions to gain more information) and exploitation (using known actions that yield high rewards).

The general procedure for an agent interacting with the environment can be described as follows:

1. The agent starts in an initial state provided by the environment, i.e., $S_t \rightarrow S_0$.
2. It selects an action A_t based on its current state S_t and a predefined policy.
3. The action is applied to the environment, leading to a state transition.
4. The environment provides feedback to the agent in the form of a reward signal.
5. The agent receives the reward R_{t+1} and observes the next state S_{t+1} resulting from the action.
6. Based on this information, the agent updates its knowledge or internal model of the environment.
7. The agent repeats this process, selecting actions, receiving rewards, and updating its knowledge, to learn and improve its decision-making abilities over time.

2.2 Deep Q learning

Q learning [25] is an algorithm that optimizes the policy of the agent by learning a Q-value, which is a measure of the quality of a state–action pair. The Q-value is denoted as $Q_\pi(S_t, A_t)$, which represents the expected sum of the discounted rewards that an agent can achieve by starting from state S_t , taking action A_t , and following π for the remaining steps. The Q-value for the same state–action pair may vary depending on the policy because different policies may lead to distinct expected returns. The Q-value is calculated using Eq. (1) with the discount factor $\gamma \in (0, 1]$.

$$Q_\pi(S_t, A_t) = \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots] \quad (1)$$

The goal of Q learning is to determine the optimal Q-value that indicates the maximum expected return achievable for a state–action pair. Q learning employs the Bellman optimality equation for Q-values, which provides a recursive decomposition of optimal Q-values that reflects the relationship between the current actions and future rewards. Equation (2) can be formulated to incorporate the maximum Q-value among all possible actions in the next state.

$$Q_\pi(S_t, A_t) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} (Q_\pi(S_{t+1}, a'))\right] \quad (2)$$

For finite state and action spaces, the Q-value can be computed using dynamic programming with the tabular method, whereas deep Q learning [26] employs a function

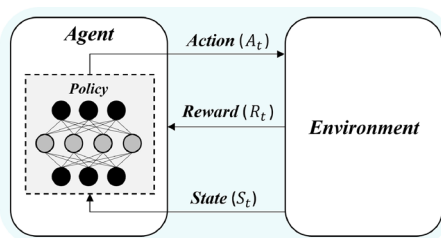


Fig. 1 The interaction between the agent and environment in DRL

approximation method, such as a deep neural network, to handle large and continuous state and action spaces, which cannot be addressed in conventional dynamic programming methods. A deep Q network (DQN) is a deep neural network with multiple layers that takes a state as the input and produces a vector of Q-values corresponding to each action as the output. The central concept of the DQN algorithm is to address scenarios with high-dimensional state and action spaces.

The ϵ -greedy policy [27] balances exploration and exploitation. Exploration involves attempting new actions to discover their potential rewards, while exploitation entails selected actions that are believed to yield the highest expected rewards based on current knowledge. The agent selects exploration with a probability of ϵ .

$$A_t = \begin{cases} \arg \max_a Q_{\text{est}}(S_t, a) & \text{with probability } 1 - \epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases} \quad (3)$$

Figure 2 depicts the deep Q learning process using Double DQN. The DQN algorithm becomes more stable and efficient when incorporating experience replay and Double DQN [28, 29]. Experience replay [30] allows the agent to learn from a diverse set of experiences and breaks the temporal correlation, while Double DQN reduces the overestimation bias in the Q-value estimation, leading to more accurate and stable learning. These techniques are crucial for improving the performance and convergence of the DQN algorithm in various DRL tasks. Double DQN comprises two neural networks: an online model with weights denoted by θ_t and a target model with weights denoted by θ'_t . The target model is periodically updated using the weights from the online model. The key concept of Double DQN is to decouple the action selection and estimation processes of the target Q-value. The online model is responsible for action

selection for the next state, whereas the target model is used to estimate the Q-value for the selected next action. This helps to mitigate overestimation bias and provides more accurate Q-value estimates. The target value is calculated using Eq. (4):

$$Y_t \equiv R_{t+1} + \gamma Q(S_{t+1}, \arg \max_a Q(S_{t+1}, \arg \max_a \theta_t; \theta'_t)). \quad (4)$$

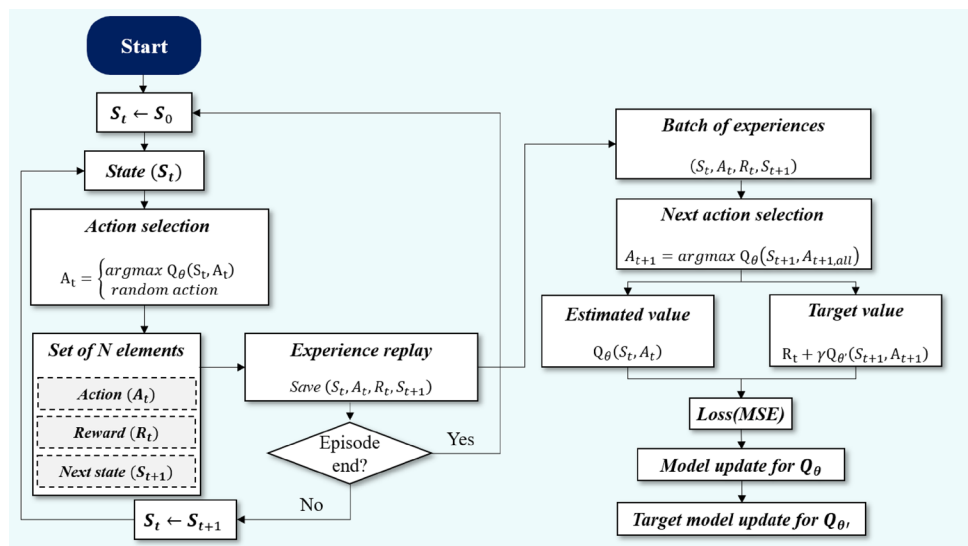
During training, the online model is updated as in the original DQN algorithm using the Bellman equation and mean squared error (MSE) loss function. In addition, the target model is updated periodically by copying the weights from the Q network. This delay in updating the target network helps to stabilize the learning process and reduce the variance in the Q-value estimation. The loss calculation is presented in Eq. (5):

$$\mathcal{L} = (Q(S_t, A_t; \theta_t) - Y_t)^2. \quad (5)$$

2.3 Prioritized experience replay

DQN training is performed by extracting the transition information (S_t, A_t, R_t, S_{t+1}) in batches from the experience replay buffer, which stores various types of transition information. When extracting batches from the buffer, the extraction of diverse and uncorrelated transition information is important. Therefore, if a probability proportional to the temporal difference error δ , which is the difference between the predicted and the target values calculated using Eq. (5), is assigned to extract a batch, important transition information can be extracted with priority [30]. This strategy is called prioritized experience replay (PER). The important transition priority p_i for each transition i is calculated using Eq. (6).

Fig. 2 Flowchart of deep Q learning using Double DQN



Therefore, for transition i , the probability P_i of sampling transition i is calculated using Eq. (6).

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \quad (6)$$

where $p_i = |\delta_i| + \varepsilon$.

2.4 Positional encoding

The positional encoding method represents the relative position of a token in a sequence as an embedding vector with dimension d_{model} to facilitate understanding of the neural network model [31]. Equation (7) shows that the elements of each vector are represented by sinusoidal values with different frequencies that are regulated by f :

$$PE_{(pos, 2i)} = \sin\left(\frac{pos}{f^{2i/d_{model}}}\right)$$

$$PE_{(pos, 2i+1)} = \cos\left(\frac{pos}{f^{2i/d_{model}}}\right), \quad (7)$$

where pos is the position to be embedded and i indicates the location of the vector component. This expression embeds the relative position information into a distinct vector.

3 Defining mesh generation as a DRL problem

In this study, the problem of boundary grid generation is redefined as the problem of determining an optimal path. That is, each node on the surface serves as an agent that learns the optimal step-by-step path towards the outer space without colliding with other nodes. It is essential not only to avoid collisions, but also to ensure that the cells generated by the step-by-step paths exhibit high grid quality.

As mentioned earlier, due to the challenging nature of finding suitable initial learning conditions in DRL, the problem definition was based on a two dimension. This approach not only minimizes the time required to find viable learning conditions but also facilitates easier interpretation of the results before extending the framework to three dimensions.

Appropriate definitions of the components of DRL, namely state, rewards, action, and environment, are necessary to tackle this redefined problem using DRL. The DRL cannot be trained properly without defining these components appropriately. Figure 3 illustrates this concept and provides a brief description of the proposed definition for each component. The movement paths taken by the nodes (acting as “agents”) to move towards the outer space are defined as “actions” (A_t). The shape of the surface, which is necessary information for determining these actions, is

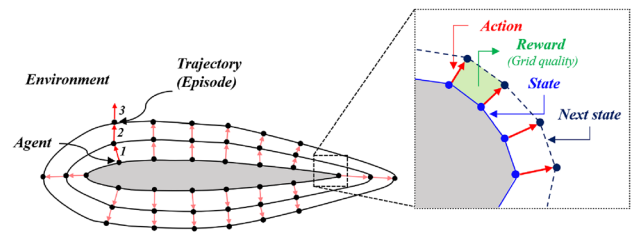


Fig. 3 The basic concept of redefining mesh generation as a DRL problem

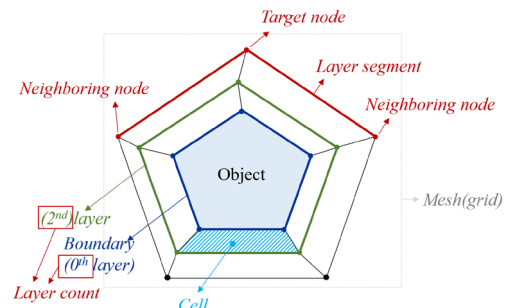


Fig. 4 Description of the terms used in the present work

defined as the “state” (S_t). The quality of the cell in the grid generated by these actions is defined as the “reward” (R_t). Finally, the “environment” is defined as a system that receives the actions of the agent, creates a new grid layer, and calculates and returns the rewards of the grid.

The remainder of this section provides a detailed explanation of each component. The terms used later are defined in Fig. 4. Terms such as “Boundary,” “Node,” “Layer,” and “Cell” are commonly used in fluid dynamics grid terminology. (Since the study deals with a two dimension problem, the term “boundary” will be used instead of “surface” hereafter.) We introduce the new terms “Layer segment” and “Layer count.” A layer segment refers to the portion of the layer consisting of the target node positioned at the center and its neighboring nodes. The layer count refers to the ordinal number of layers taking action, assuming that the boundary is the 0th layer. The layer count also serves as a time step for the main episode in this study.

3.1 State members

The state encompasses all parameters of the environment that the agent must consider before taking action. The most crucial factor in determining the movement of nodes is the shape of the layer in which the node is located. Ideally, having information regarding all nodes contained in the layer is advantageous for taking action. However, including

information regarding all nodes in the layer is not a good option for two reasons.

First, a layer of a CFD grid contains a vast number of nodes, ranging from hundreds to tens of thousands or more. Therefore, considering information regarding all nodes for a single action is highly inefficient.

Second, to conduct DRL, a network must be trained that receives the inputs of a fixed-length vector. Therefore, training an algorithm by including information regarding all nodes in the input would limit its use in boundary grids containing the same number of nodes. This severely diminishes the effectiveness of the algorithm.

Therefore, we propose a layer segment denoted by $\mathbf{p}_t^i$, as defined by Eq. (8), as the first state member. The target node (p_t^i) is the node taking action and the remaining m neighboring nodes on both sides convey information regarding the shape of the boundary to the target node. The parameter t represents the layer count. Figure 4 depicts the layer segment when $m = 1$.

$$\mathbf{p}_t^i = (p_t^{i-m}, \dots, p_t^{i-1}, p_t^i, p_t^{i+1}, \dots, p_t^{i+m}) \quad (8)$$

The second state member is the layer count. The optimal action may vary depending on the number of current layers being built, even if the positions of the surrounding nodes are the same. Here, the layer segment information consists of a vector of length $2m + 1$ composed of x, y coordinate pairs, but the layer count is a scalar value. Therefore, when used together, an information imbalance exists. To address this imbalance, the layer count is vectorized using positional encoding. The vectorized layer count, denoted by $\mathbf{q}_t^i$, is obtained by substituting the layer count into the pos variable in Eq. (7). If a total of N nodes exist within one layer with a layer count t , the set of all states within one layer can be expressed as follows:

$$S_t = \{S_t^1, S_t^2, \dots, S_t^i, \dots, S_t^{N-1}, S_t^N\} \text{ where } S_t^i = \{\mathbf{p}_t^i, \mathbf{q}_t^i\}. \quad (9)$$

3.2 Action members

The path taken by the target node to create the next layer is defined as a combination of two components, as shown in Fig. 5. The first component, the acting distance, is the Euclidean distance between the current and next positions of the target node. The second component, the acting angle, is the angle at which the target node moves and is defined as the angle when the direction bisecting $\angle p_t^{i-1} p_t^i p_t^{i+1}$ is considered to be 0 degrees.

In general, the height of the grid decreases as it approaches the boundary and increases up to a certain size as it moves away from the boundary. Therefore, if the fixed values of acting distance is defined as an action, it should cover a very wide range. This is not a good choice for

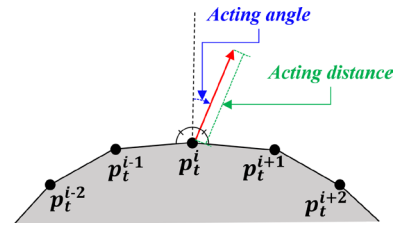


Fig. 5 Definition of the action space

defining actions because the learning efficiency decreases as the number of actions increases. Therefore, instead of the acting distance, the axial action coefficient (C_{ax}) is defined as the action. The acting distance using C_{ax} is defined by Eq. (10). C_{ax} is the ratio between the actual displacement distance and reference length, which is the product of the height of the first cell (h) and the power of the growth rate (r). In this case, t represents the layer count.

$$\text{Acting distance} = C_{ax} \cdot h \cdot r^{t-1} \quad (10)$$

The range of acting angles that the target node can take depends on the appearance of the layer segment. If $\angle p_t^{i-1} p_t^i p_t^{i+1}$ is small, the movement occurs in a very narrow area, and if $\angle p_t^{i-1} p_t^i p_t^{i+1}$ is large, it moves in a larger area. If the fixed values of acting angle is defined as an action, it may select actions that invade the previous layer when $\angle p_t^{i-1} p_t^i p_t^{i+1}$ is small, or some areas may not be covered when $\angle p_t^{i-1} p_t^i p_t^{i+1}$ is large. Therefore, instead of the acting angle, we propose the radial action coefficient (C_{rad}) as the action. The definition of the acting angle using C_{rad} is provided in Eq. (11), where C_{rad} represents the ratio of the acting degree to $\angle p_t^{i-1} p_t^i p_t^{i+1}$.

$$\text{Acting angle} = C_{rad} \cdot \angle p_t^{i-1} p_t^i p_t^{i+1} \quad (11)$$

3.3 Reward functions

The reward is a score that evaluates whether or the action performed by the agent is good. In grid problems, it is rational for the grid quality to serve as the reward. The quality of the grid can be evaluated both from a geometric perspective and in conjunction with CFD. However, as CFD analysis results were not considered in this study, the quality of the grid was evaluated purely from a geometric perspective.

The geometric quality of the grid is mainly determined by how close the grid is to being rectangular (orthogonality) and how smoothly the grid shape changes (smoothness), which can be evaluated using the skewness and Jacobian metrics [32]. In this study, a length ratio [33] was added to these two metrics to evaluate the smoothness over a broader range, thereby defining the reward function.

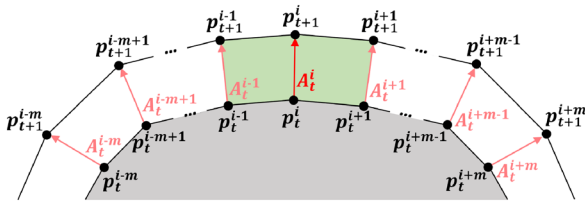


Fig. 6 Mesh cells generated at nodes on the t -th layer upon execution of action (A_t)

The original definition of skewness is expressed by Eq. (13) and is represented by the four angles inside the cell (Θ). When the largest angle among the angles of the cell is closer to 180 degrees and the smallest angle is closer to 0 degrees, the value approaches 0 more closely. A negative value is calculated if any angle exceeds 180 degrees. A maximum value of 1 is calculated only when all four angles are 90 degrees. As shown in Fig. 6, because two cells are directly influenced by a single action, the skewness of these two cells is averaged to form Eq. (12), which is the definition of skewness in this study.

$$\text{Skewness} = \frac{S(\Theta_1) + S(\Theta_2)}{2} \quad (12)$$

$$\text{where } S(\Theta) = 1 - \max\left(\frac{\max(\Theta) - 90^\circ}{90^\circ}, \frac{90^\circ - \min(\Theta)}{90^\circ}\right) \quad (13)$$

$$\Theta_1 = \{\angle p_{t+1}^i p_{t+1}^{i-1} p_{t+1}^{i+1}, \angle p_{t+1}^{i-1} p_{t+1}^i p_{t+1}^{i+1}, \angle p_{t+1}^{i+1} p_{t+1}^i p_{t+1}^{i-1}, \angle p_{t+1}^i p_{t+1}^{i-1} p_{t+1}^{i+1}\} \quad (14)$$

$$\Theta_2 = \{\angle p_{t+1}^{i+1} p_{t+1}^i p_{t+1}^{i+1}, \angle p_{t+1}^i p_{t+1}^{i+1} p_{t+1}^{i+1}, \angle p_{t+1}^{i+1} p_{t+1}^i p_{t+1}^{i+1}, \angle p_{t+1}^i p_{t+1}^{i+1} p_{t+1}^{i+1}\} \quad (15)$$

The definition of the Jacobian is presented in Eq. (16) and is defined as the ratio of the determinants of the two cells that are influenced by a single action. This serves as a measure of the similarity of the areas of adjacent cells, such that if the shapes of the two cells are identical, the maximum value of 1 is achieved. However, even if the areas of the two cells are the same, a low value can be obtained if there is a significant difference in the shape.

$$\text{Jacobian} = \frac{\min[\det((p_{t+1}^i p_{t+1}^{i-1}, p_{t+1}^i p_{t+1}^{i+1})), \det((p_{t+1}^{i+1} p_{t+1}^i, p_{t+1}^{i+1} p_{t+1}^{i+1}))]}{\max[\det((p_{t+1}^i p_{t+1}^{i-1}, p_{t+1}^i p_{t+1}^i)), \det((p_{t+1}^{i+1} p_{t+1}^i, p_{t+1}^{i+1} p_{t+1}^{i+1}))]} \quad (16)$$

The length ratio is defined by Eq. (17). First, as in Eq. (18), the average length of the upper side of m cells on both sides of the target node is calculated. Subsequently, the ratios of the lengths of the upper sides of the two cells created by the action to this average length are obtained. The equation is constructed such that the value approaches zero as the

difference between the two target lengths (Eq. (19)) and the average length increases. The length ratio takes the smaller value between these two values.

$$\text{Length ratio} = \min\left(\frac{\min(l_{\text{ref}})}{l_{\text{nb}}}, \frac{l_{\text{nb}}}{\max(l_{\text{ref}})}\right) \quad (17)$$

$$\text{where } l_{\text{ref}} = (\|p_t^{i-1} p_t^i\|, \|p_t^i p_t^{i+1}\|) \quad (18)$$

$$l_{\text{nb}} = \mathbb{E}(\|p_t^{i-m} p_t^{i-m+1}\|, \dots, \|p_t^{i-2} p_t^{i-1}\|, \|p_t^{i+1} p_t^{i+2}\|, \dots, \|p_t^{i+m-1} p_t^{i+m}\|) \quad (19)$$

A reward function is proposed by combining these three metrics to calculate the mesh quality, as shown in Eq. (20). In this case, w_{skew} , w_{length} , and w_{jacob} represent the respective weights for each metric, with their sum being 1. Therefore, the range of the final reward is 0 to 10.

$$\begin{aligned} \text{Reward} = & (w_{\text{skew}} \cdot \text{Skewness} + w_{\text{length}} \cdot \\ & \text{Length ratio} + w_{\text{jacob}} \cdot \text{Jacobian})^2 \cdot 10 \end{aligned} \quad (20)$$

where $w_{\text{skew}} + w_{\text{length}} + w_{\text{jacob}} = 1$

In addition to the mesh quality, an important consideration is the negative volume, which denotes improperly generated grids. As depicted in Fig. 7, when two actions cross, a negative volume occurs, making the grid unusable. A negative volume is the most significant issue in grid generation, and once it occurs, grid generation is immediately terminated. When the grid generation is halted, it affects the overall acquired Q-value, leading to learning that naturally avoids crossed actions. However, in this case, a penalty policy is added to regulate such actions more strongly.

When the actions A_t^i and A_t^{i+1} are entangled, it is evident that the rewards R_t^i and R_t^{i+1} should incur penalties. However, even if the actions of the nodes surrounding these two actions; that

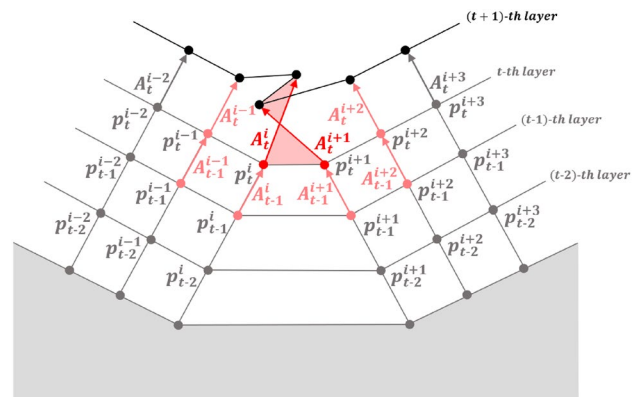


Fig. 7 Entangling between adjacent actions

is, the actions on both sides (A_t^{i-1} and A_t^{i+2}) and in the $(t-1)$ -th layer (A_{t-1}^{i-1} , A_{t-1}^{i+1} , A_{t-1}^{i+2} , and A_{t-1}^{i+3}), do not result in a crossing, they still need to receive some penalty. This is because they contribute to the crossing of two actions. The number of nodes that should share the penalty for crossed actions may vary depending on the problem. In this study, we propose imposing penalties on the nodes that take the crossed action, as well as on the two nodes on each side of these nodes and their immediate predecessors (nodes at the $(t-1)$ -th layer). Therefore, a total of 12 nodes receive penalties when a crossed action occurs.

3.4 Target model update strategies

The target model is periodically updated to stabilize and improve the learning process. Two methods are available for updating the target model in Double DQN. In a hard update, the target model is updated at fixed intervals: every N training steps. As shown in Eq. (21), the weights of the online model are copied directly to the target model.

$$\theta'_t \leftarrow \theta_t \quad (21)$$

In a soft update, the target model is updated continuously: at each training step. As shown in Eq. (22), a fraction of the weights of the online model is blended into the target model. The fraction is controlled by a hyperparameter denoted as “ τ ,” which is typically set to a small value.

$$\theta'_t \leftarrow (1 - \tau)\theta'_t + \tau\theta_t \quad (22)$$

3.5 Neural network training strategies

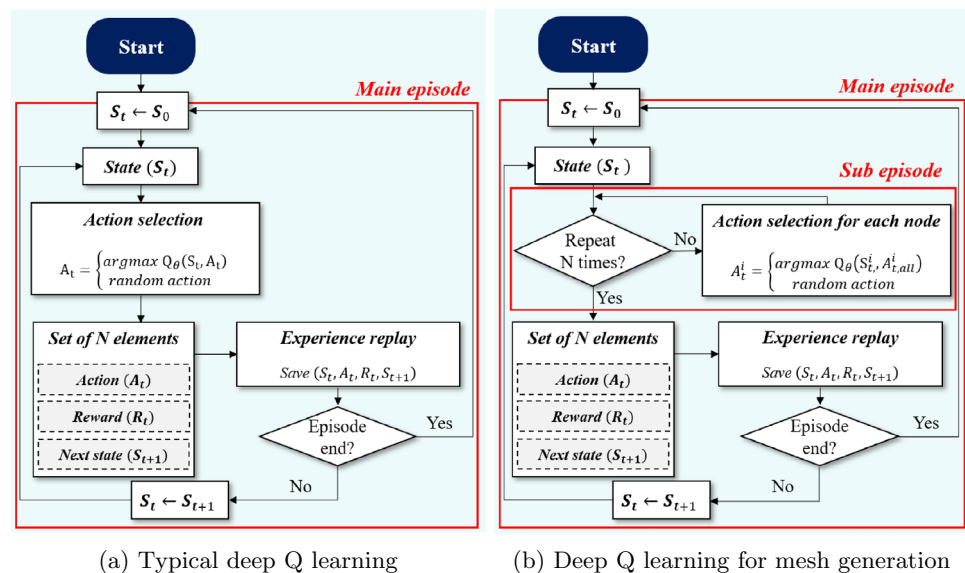
Computing the loss is necessary to train a neural network. In DRL, the loss is the difference between the target value and predicted Q-value. In this study, two methods for computing this difference and utilizing it for neural network training are proposed and compared. The first method is layer-wise training. This involves averaging the loss for each layer (Eq. (23)) after determining the action for all nodes within a layer, thereby updating the network. The second method is node-wise training. This involves averaging the loss for nodes with the same index (Eq. (24)) to update the network. In the equations, B represents the batch size, N is the number of nodes per layer, and Y_t^i is the target value defined in Eq. (4).

$$\mathcal{L} = \frac{1}{B} \sum \left(\frac{1}{N} \sum_{i=1}^N (Q(S_t^i, A_t^i; \theta_t) - Y_t^i)^2 \right) \quad (23)$$

$$\mathcal{L}^i = \frac{1}{B} \sum (Q(S_t^i, A_t^i; \theta_t) - Y_t^i)^2 \quad (24)$$

In the first method, because the loss is computed every time a single time step progresses or the layer count increases, updating the weights at each time step appears reasonable. In the second method, because all nodes should contribute equally to the training, each training step undergoes N updates for the N nodes in a layer. Hence, performing weight updates once per episode, after accumulating a certain amount of data, is reasonable. The following algorithm outlines all of these processes in pseudocode.

Fig. 8 Comparison of the experience acquisition process of deep Q learning in the mesh generation process



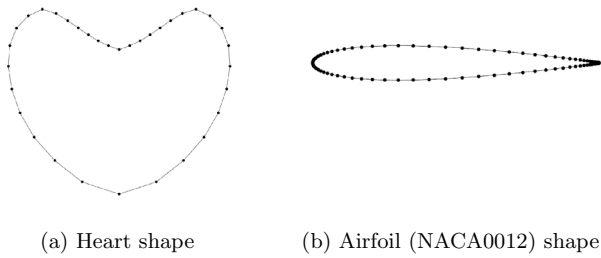


Fig. 9 Boundary grids used in the training

Table 1 Common parameters for training environment

Parameter	Value
First layer height (h)	0.005
Layer growth rate (r)	1.2
Total number of layers (t_{max})	20
Reward weights $\{w_{skew}, w_{length}, w_{jacob}\}$	$\{0.5, 0.2, 0.3\}$
Axial action coefficients (C_{ax})	$\{0.75, 1.0, 1.25, 1.5, 1.75, 2.0, 2.25\}$
Radial action coefficients (C_{rad})	$\{-\frac{2}{12}, -\frac{1}{12}, 0, \frac{1}{12}, \frac{2}{12}\}$
Penalty $\{t - th \text{ layer}, (t - 1) - th \text{ layer}\}$	$\{-20, -10\}$

Algorithm 1 Agent-based mesh generator

```

Initialize network parameters  $\theta_t, \theta_{t'} \leftarrow \theta_t$ 
1: for each episode do
2:   Get initial state  $S_0$ 
3:   for each time step do
4:     for each node do
5:       Select action  $A_t^i \sim \pi_{\theta_t}(S_t^i)$ 
6:       Observe reward  $R_t^i$  and new state  $S_{t+1}^i$ 
7:     end for
8:     Store  $(S_t, A_t, R_t, S_{t+1})$  in replay buffer
9:     if crossed then
10:      break
11:    end if
12:    if Layer-wise training then
13:      Sample  $B$  batches of  $(S_t, A_t, R_t, S_{t+1})$  from replay buffer
14:      Calculate  $\mathcal{L}$ 
15:      Update online model  $\theta_t \leftarrow \theta_t - \lambda \nabla_{\theta_t} \mathcal{L}$ 
16:    end if
17:    if soft update then
18:      Update target model  $\theta_{t'} \leftarrow (1 - \tau)\theta_t + \tau\theta_{t'}$ 
19:    end if
20:  end for
21:  if Node-wise training then
22:    Sample  $B$  batches of  $(S_t, A_t, R_t, S_{t+1})$  from replay buffer
23:    for each node do
24:      Calculate  $\mathcal{L}^i$ 
25:      Update online model  $\theta_t \leftarrow \theta_t - \lambda \nabla_{\theta_t} \mathcal{L}^i$ 
26:    end for
27:  end if
28:  if hard update then
29:    Update target model  $\theta_{t'} \leftarrow \theta_t$ 
30:  end if
31: end for

```

3.6 Sub-episodes and training process

As described in Sect. 1, the grid generation problem has many features that are suitable for the formulation of DRL. Nonetheless, a significant challenge arises because the reward for the action of a single node is not calculated deterministically. This is because evaluating the reward requires determining the actions for neighboring nodes as well as for the target node. To address this issue, we introduce the concept of sub-episodes. While the main episode represents the process of stacking grids layer by layer from the boundary to the final layer, the sub-episode entails traversing nodes within a single layer. The reward for all nodes can be calculated deterministically by traversing all nodes in a layer and determining their actions.

The experience acquisition processes in DRL, which is the typical method, and the proposed method are compared in Fig. 8. In typical DRL, because rewards for actions can be obtained directly, experience replay is accumulated for each time step. However, in the proposed process, rewards for all actions are obtained only after completing a sub-episode and experience replay is constructed and stored thereafter.

4 Results and discussion

Using the DRL problem definition in Sect. 3 as a basis, training was conducted to generate the two dimensional boundary layer grids. Initially, the boundary grid for training was selected. Figure 9 shows the two boundary grids used in this study. The heart-shaped boundary has 34 nodes, whereas the airfoil (NACA0012) has 101. In general, generating a boundary mesh for boundaries with concave parts is more challenging than that for boundaries with convex parts. Furthermore, the large number of boundary nodes in the proposed framework implies that a large number of agents must be trained. Therefore, to determine the optimal conditions for

grid generation, selecting a shape with fewer nodes but that contains segments of various shapes is efficient. Therefore, a case study on core learning strategies and parameters was conducted for the heart shape, which contains concave segments with fewer nodes. Finally, the optimal learning conditions derived from the heart-shaped boundary grid were applied to an airfoil shape with a large number of nodes to confirm their effectiveness on other boundary grids.

Several parameters commonly used in training were determined first, followed by a case study focusing on the training strategies and parameters based on these settings. Table 1 lists the parameters commonly employed in all training runs described in this section. The height of the first layer was set to 0.005, which corresponds to a dimensionless number equal to half of the maximum width of the heart shape and the chord length of the airfoil shape. The growth rate for layer advancement was 1.2, and the agents determined whether to move less or more than the reference layer advancement length that was calculated based on this value. The final number of layers was set to 20, which is a commonly used value in the boundary layer grid generation for CFD.

In general, a boundary layer mesh refers to a grid that is densely stacked near the boundary to capture rapid variations in the flow field. However, in this study, the default parameters were set to generate grids that were much thicker than the typical boundary layer grids by increasing the height of the layers to enhance the visibility of the learning results.

The weights for each grid quality metric (skewness, length ratio, and Jacobian) that were used to calculate the reward values were 0.5, 0.2, and 0.3, respectively. The action coefficient determines the distance for actual agent movements in the axial and radial directions. The actual axial movement (acting distance) was calculated using Eq. (10) and the radial movement (acting angle) was calculated using Eq. (11). With seven coefficients selected for the axial action and five coefficients selected for the radial action, the total action space consisted of 35 actions. Finally, a penalty of -20 was assigned to the layer where crossing occurred (t -th layer) and a penalty of -10 was assigned to the preceding layer ($(t - 1) - th$ layer).

Table 2 lists the parameters of the neural network used in the training. Fully connected dense layers with three layers

Table 2 Parameters for neural network

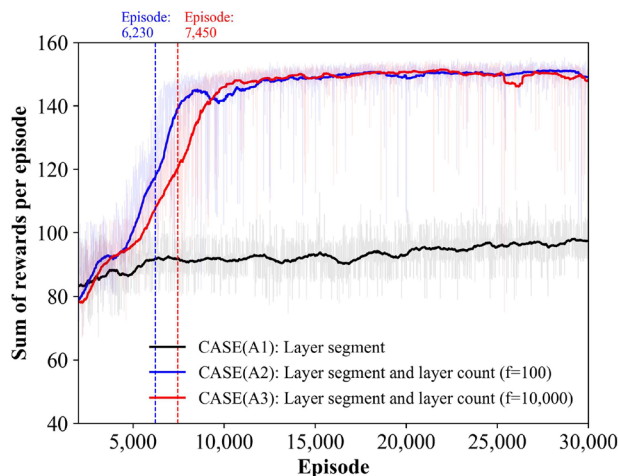
Parameter	Value
Layer type	Fully connected layer
Number of hidden layers	3
Number of hidden nodes	256
Activation function	GeLU
Optimizer	Adam
Loss function	MSE
Learning rate	0.001
Batch size	128
Buffer size	10,000

Table 3 Proposed learning conditions for mesh generation

No	Parameter	Value
1	Definition of state (S_t^i)	$\{\mathbf{p}_t^i, \mathbf{q}_t^i\}$ ($f = 100$)
2	State normalization	2 pts method
3	Number of neighboring nodes (m)	5
4	Training strategy	Node-wise training
5	Target model update method	Soft update

Table 4 Case descriptions according to state definition

	S_t^i	f	Proposed
Case (A1)	$\{p_t^i\}$	-	
Case (A2)	$\{p_t^i, q_t^i\}$	100	*
Case (A3)	$\{p_t^i, q_t^i\}$	10,000	

**Fig. 10** Reward sums of meshes generated by the network trained using different state members

and 256 nodes each were used. The GeLU activation function was applied to each layer except for the final layer. The Adam optimizer and MSE loss function were employed. The learning rate was set to 0.001 and the batch size was set to 128. Another parameter used in the training was the replay memory size or buffer size, which was set to 10,000.

Before presenting the detailed case study results, the five proposed optimal training strategies and parameter settings are summarized in Table 3. The following parts discuss the ablation study for each strategy and parameter based on the settings in Table 3.

Although the introduction of sub-episodes resolves the issue of non-deterministic rewards, two challenges remain. The first is the high dimensionality of the agents and states. To address this, we propose using lower-dimensional layer segments as states instead of leveraging the high-dimensional information of the entire node. However, if the amount of usable information is reduced excessively, learning may not proceed effectively, necessitating further studies on additional state members, namely the layer count, and the size of the layer segments. In addition, further studies are required to normalize the node positions to reduce the learning space for training a large number of agents effectively.

The second challenge involves the dependencies between actions. The actions of closely located nodes are highly related; therefore, even if a target node performs a good

action, it may not receive a good reward if the neighboring nodes perform bad actions. In this study, as mentioned in Sect. 3.2, we address this issue by defining penalties for actions that result in a negative volume. However, relying solely on penalty policies may make training models for problems involving high dependencies between actions difficult. Establishing a deliberate training strategy to deliver rewards for actions to the network effectively is necessary.

The first to third proposed learning conditions aim to tackle the first challenge: the high dimensionality of the action and state problem. Detailed discussions of this issue are presented in Sects. 4.1, 4.2, and 4.3. The fourth and fifth proposed learning conditions aim to tackle the second challenge: the action dependency problem. Detailed discussions of this aspect are presented in Sects. 4.4 and 4.5.

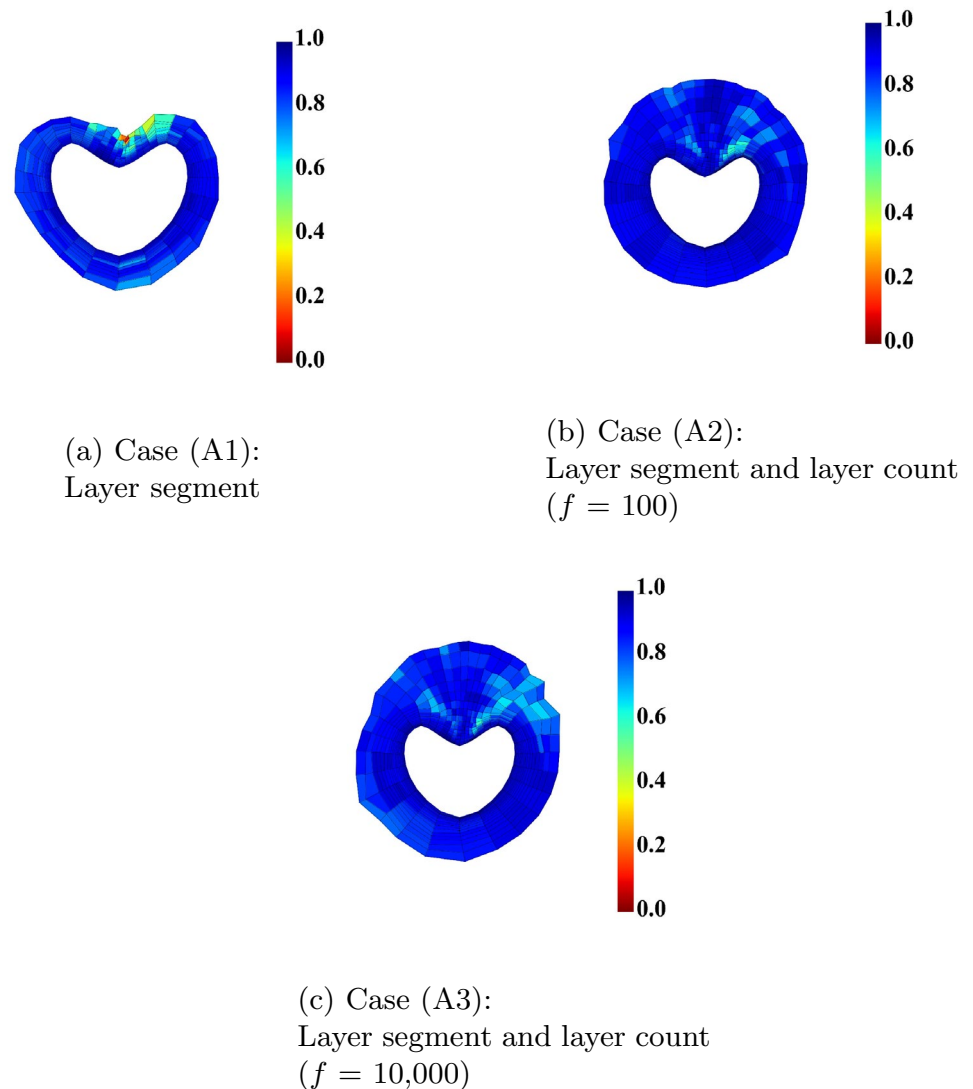
4.1 Study on state members

As mentioned previously, the layer segment p_t^i and layer count q_t^i are defined as state members. First, the results are compared based on the composition of the state members.

Table 4 presents descriptions of the training scenarios based on the state members. Three cases were compared: one using only layer segments as the state and the others incorporating layer count information. The layer segment information consists of vectors with a length of $2m - 1$, containing pairs of x and y coordinates. Conversely, the layer count is represented as a scalar value ranging from 1 to 20, which leads to an imbalance in the information when combined. To address this imbalance, positional encoding was employed to vectorize the layer count. During the vectorization process, the structure of the transformed information varies according to the period of the trigonometric function. The parameter controlling this variation, which is denoted by f in Eq. (7), increases the period of the trigonometric function referenced during transformation, thereby reducing discrepancies between instances of the transformed vectors. In [31], a value of 10,000 was used for f to encode the positional information within sentences. Therefore, we propose a value of 100 for f . The length of the transformed vector (d_{model} in Eq. (7)) was set to 22 to match the length of the layer segment.

Figure 10 shows the sum of the averaged rewards of the cells in each layer generated within an episode obtained from the mesh generated using the network that was updated as the episodes progressed. Owing to significant fluctuations across episodes, the graph is represented as a moving average over 100 episodes. The values for each episode are shaded. The boundary grid used for training comprised 34 nodes, with the rewards per node ranging from 0 to 10. By averaging the rewards across each layer and summing them for 20 layers, an ideal sum of up to 200 rewards per episode could be achieved. However, even with a good grid,

Fig. 11 Quality of the grid generated by the network trained using different state members



the actual value typically reached approximately 160. This difference was owing to the need for flattened grids of lower quality near the boundary and skewed grids around curved boundary areas to avoid a negative volume.

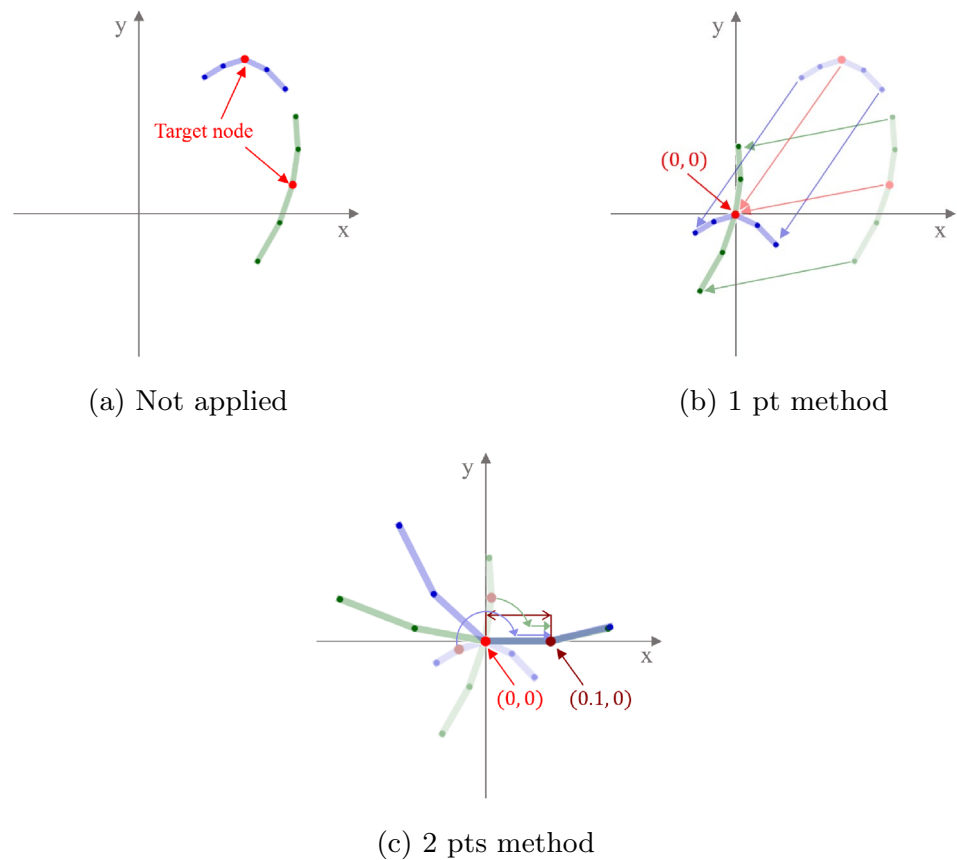
It is evident from the results that in cases (A2) and (A3), where the layer count was included in the state, a reward value indicating the possibility of attainment of high-quality grids was reached around the 3230-th and 7450-th episodes, respectively. However, in case (A1), where the layer count was not included in the state, learning did not progress properly, even up to the 30,000-th episode. This suggests that including the layer count in the state is a crucial factor for learning.

Figure 11 shows the grids generated using the networks of the episode with the highest reward from each case. As mentioned previously, substantially thicker grids were generated compared with typical boundary grids to ensure that the grid was properly generated in the concave regions through

learning. A color map is used to indicate the reward value associated with each cell. In cases in which the layer count was not used as a state member, it is evident that the grids failed to be generated properly, particularly on concave boundaries, resulting in collisions and significantly lower rewards, and they could not stack up to 20 layers. In contrast, when the layer count was used, both cases successfully stacked grids of up to 20 layers with similar grid qualities. However, employing the proposed f value of 100 yielded a slightly better distribution of rewards.

4.2 Study on node position normalization

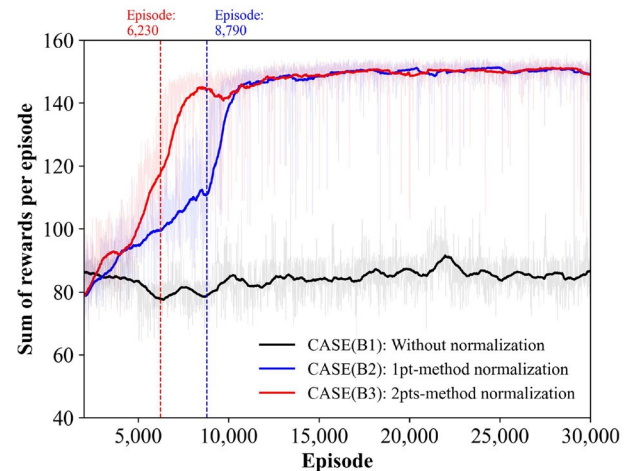
When generating grids, the absolute positions of the grid points are not crucial; rather, their positions relative to the neighboring nodes hold significance. In the framework of this study, employing normalized positions instead of absolute positions is proposed when defining a layer segment as

Fig. 12 Node position normalization methods**Table 5** Case description according to the node position normalization method

	Normalization method	No. of nodes in a layer segment	Proposed
Case (B1)	–	$2m + 1$	
Case (B2)	1 pt method	$2m$	
Case (B3)	2 pts method	$2m - 1$	*

a state member. Two methods were tested to normalize the node positions, as illustrated in Fig. 12. The first method, shown in Fig. 12b, involves linearly shifting the target node to (0,0), which is named as the 1 pt method. Using the unnormalized node positions results in $2m + 1$ nodes being included in the layer segment. However, when using the 1 pt method, all target nodes in the states are uniformly shifted to (0,0), enabling their exclusion from the state. Therefore, when employing the 1 pt method, the number of nodes used as the layer segment was reduced to $2m$.

The second method, shown in Fig. 12c, is named as the 2 pts method, where the target node (P_t^i) is moved to (0, 0), and then the subsequent node (P_t^{i+1}) after the target node is rotated and scaled to be located at (0.1, 0). That is, this method ensures that the distance between the target

**Fig. 13** Reward sums of meshes generated by the network trained using varying normalization methods

and subsequent nodes becomes the reference distance (0.1) when the target node is aligned with (0,0). By employing this method, two points, namely the target node and subsequent node, can be excluded from the state, thereby reducing the number of nodes used as the state to $2m - 1$. Normalization aligns the layer segments to a certain

Fig. 14 Quality of the grids generated by the network trained using varying normalization methods

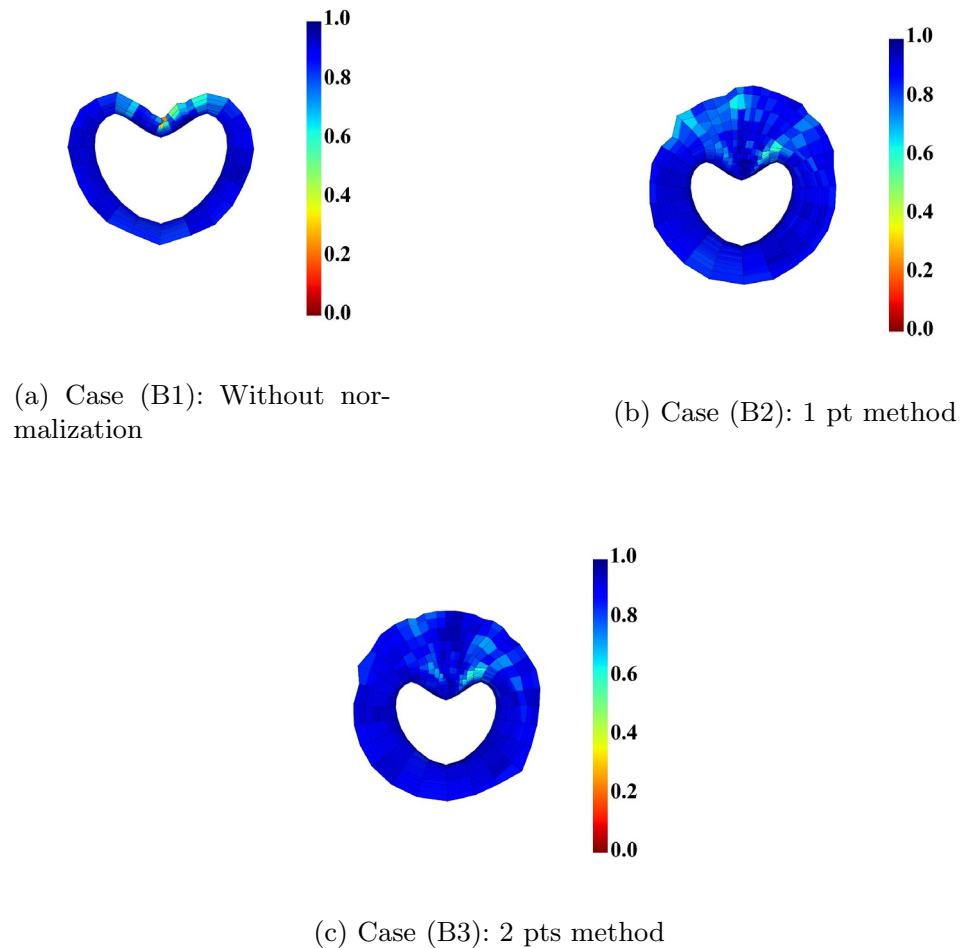


Table 6 Case descriptions according to the number of neighboring points

	No. of neighboring points (m)	Proposed
Case (C1)	2	
Case (C2)	3	
Case (C3)	4	
Case (C4)	5	*
Case (C5)	6	
Case (C6)	7	
Case (C7)	10	

extent, thereby reducing the state space and mitigating the challenges in training many agents.

Table 5 presents the descriptions of the cases based on the normalization method for the node positions. Figure 13 depicts the learning process for the cases listed in the table. As before, the y-axis represents the sum of the average rewards of the cells in each layer generated within an episode. It is evident from this graph that proper training did not progress if the node positions were not normalized. That is,

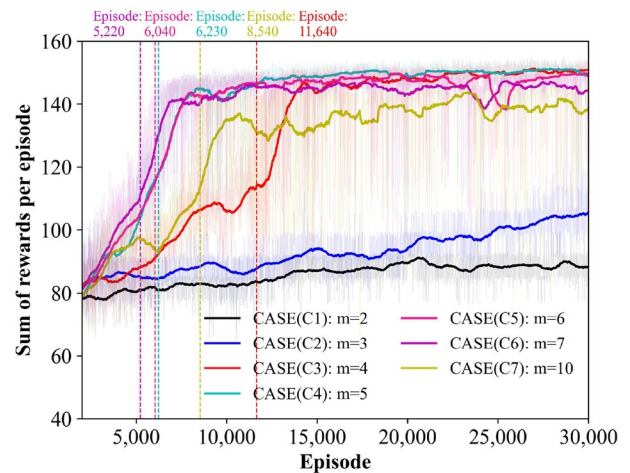


Fig. 15 Reward sums of meshes generated by the network trained using varying neighboring points

normalization to reduce the size of the state space is crucial for learning. Among the normalization methods, the 2 pts method was more effective than the 1 pt method. When using

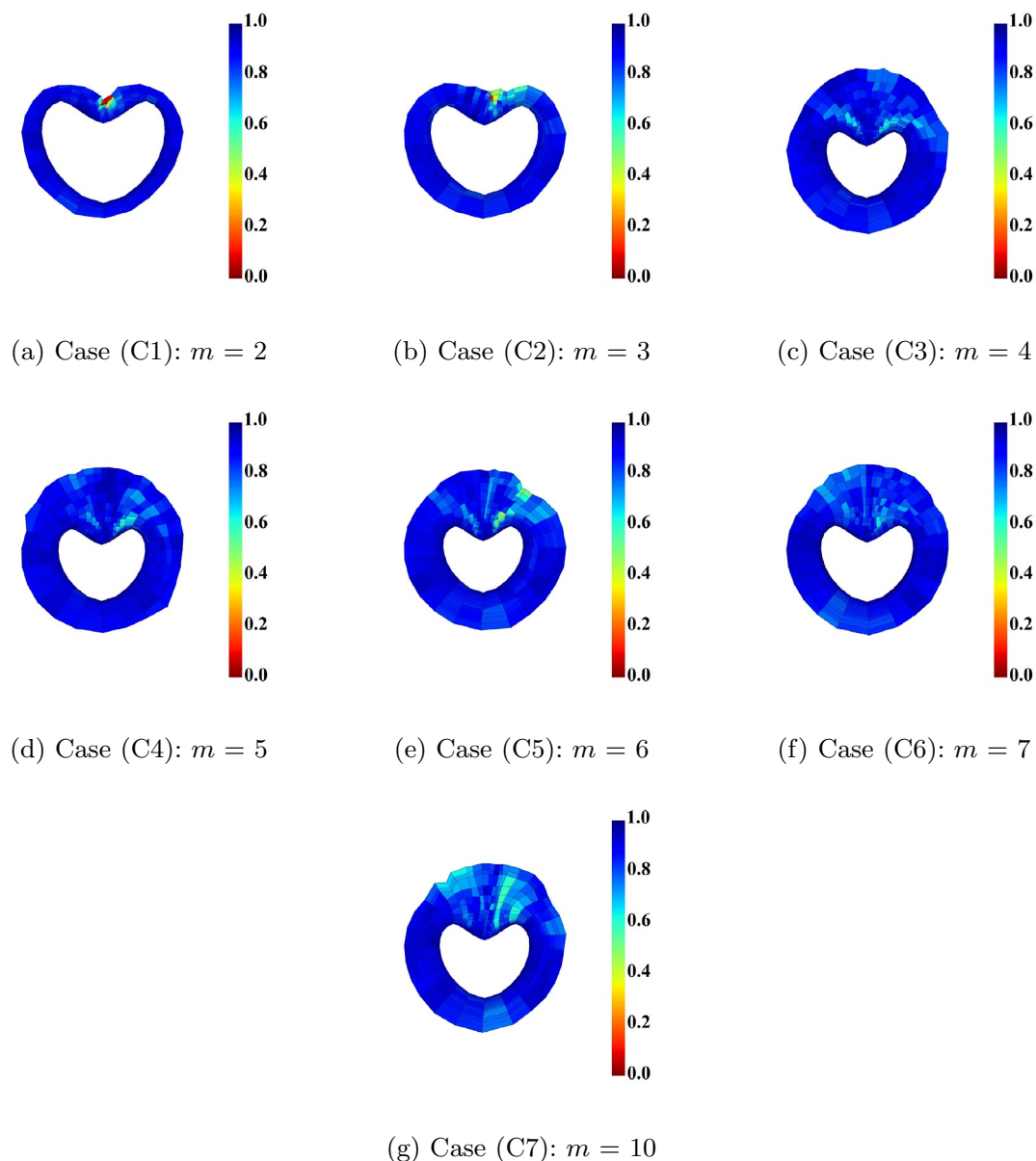


Fig. 16 Quality of the grid generated by the network trained using varying neighboring points

the 2 pts method, 20 layers were reached around the 6230-th episode, whereas employing the 1 pt method resulted in 20 layers being reached around the 8790-th episode.

Figure 14 shows the grids generated using the network from the episode with the best reward for each case. Similar to the results in the previous section, in case (B1), where training did not proceed properly, crossings occurred near the concave regions and the network failed to learn actions to resolve this issue, resulting in the inability to stack 20 layers, even up to the 30,000-th episode. In the remaining two cases, 20 layers were successfully stacked. The colormap indicates that using the 2 pts method resulted in the generation of higher-quality grids.

Table 7 Case descriptions according to training strategy

	Update strategy	Update frequency	Proposed
Case (D1)	Node-wise training	1 episode	*
Case (D2)	Layer-wise training	1 time step	

4.3 Study on number of neighboring points

A layer segment consists of m nodes on either side of the target node. Ideally, having information regarding all nodes on the boundary would be most beneficial for creating an optimal grid. However, this leads to an extensive state

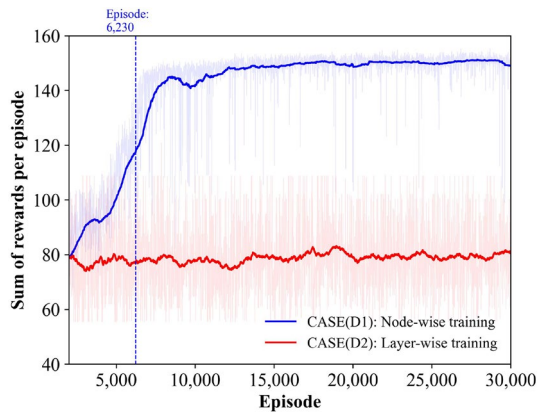


Fig. 17 Reward sums of meshes generated by the network trained using varying training strategies

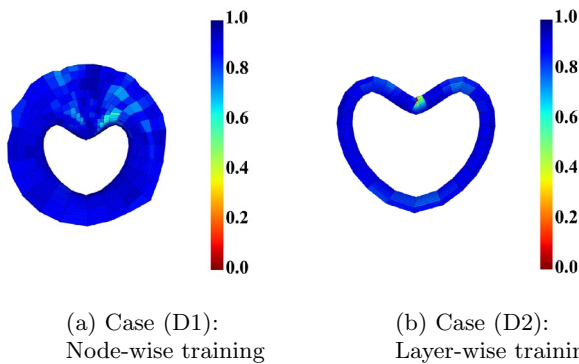


Fig. 18 Quality of the grids generated by the network trained using varying training strategies

space that potentially hinders proper learning. Conversely, a smaller value of m results in decisions based solely on the information from a small local area when selecting actions. Therefore, investigating the most effective setting for m is crucial for successful learning.

Table 6 presents descriptions of the cases based on the value of m . Figure 15 depicts the learning progress for these cases as shown in the table. As before, the y-axis represents the sum of the average rewards of the cells in each layer generated within an episode. It can be observed that proper learning did not occur when m was equal to 2 or 3. Alternatively, for $m = 3$, learning appeared to progress very slowly. As m increased to 4 or higher, reaching the maximum layers became feasible, and similar learning speeds and reward values were observed for $m = 5, 6$, and 7. However, when m reached 10, the learning speed decreased and the maximum reward value decreased. This indicates that increasing m does not necessarily benefit learning; rather, it must be set to an appropriate value (in this case, 5) for effective learning.

Table 8 Case descriptions according to the target model update strategy

	Update method	Update frequency	PER	Proposed
Case (E1)	Soft	1 time step	X	*
Case (E2)	Hard	10 episodes	X	
Case (E3)	Hard	10 episodes	O	

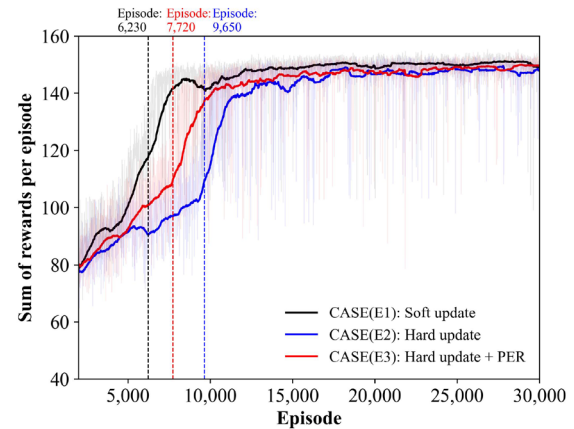


Fig. 19 Reward sums of meshes generated by the network trained using varying model update methods

Figure 16 shows the grids generated using the networks from the episodes with the highest reward in each case. Similar to the results in the previous section, cases (C1) and (C2) exhibited unsuccessful learning, with crossings occurring near the concave regions and showing low rewards. Cases (C3) to (C6) successfully stacked grids of up to 20 layers, with the best grid quality observed when using the proposed $m = 5$. In case (C7), a decrease in the reward was observed again near the concave areas.

4.4 Study on neural network training strategy

The key difference between the proposed framework and the typical DRL process is the presence of another loop within the learning loop, which traverses all nodes within a layer. This is because rewards can only be computed once the next states of all nodes are determined. Therefore, while the unit of learning in this framework could be a layer segment or target node that forms the input to the network, in terms of completeness for defining rewards, it could also be the entire layer. Hence, we investigated two learning strategies based on updating the network using the average loss computed from the layer or from nodes. Table 7 presents descriptions of the learning strategies and corresponding cases. As mentioned in Sect. 3.4, the node-wise training process was

performed once per episode, whereas the layer-wise training process was executed once per time step.

Figure 17 shows the learning progress for the cases listed in Table 7. As before, the y-axis represents the sum of the average rewards of the cells in each layer generated within an episode. It can be observed that the layer-wise training did not proceed effectively. In contrast, the node-wise training was successful. As mentioned previously, because all states and actions are complete at the layer level, layer-wise training is expected to be an effective strategy. However, the results did not support this hypothesis. Therefore, we propose selecting batches composed of layer units, reconstructing the layer segments for each target node, calculating the losses for each node using the same index, and updating the model based on their averages. Using this approach, we confirmed the feasibility of the learning.

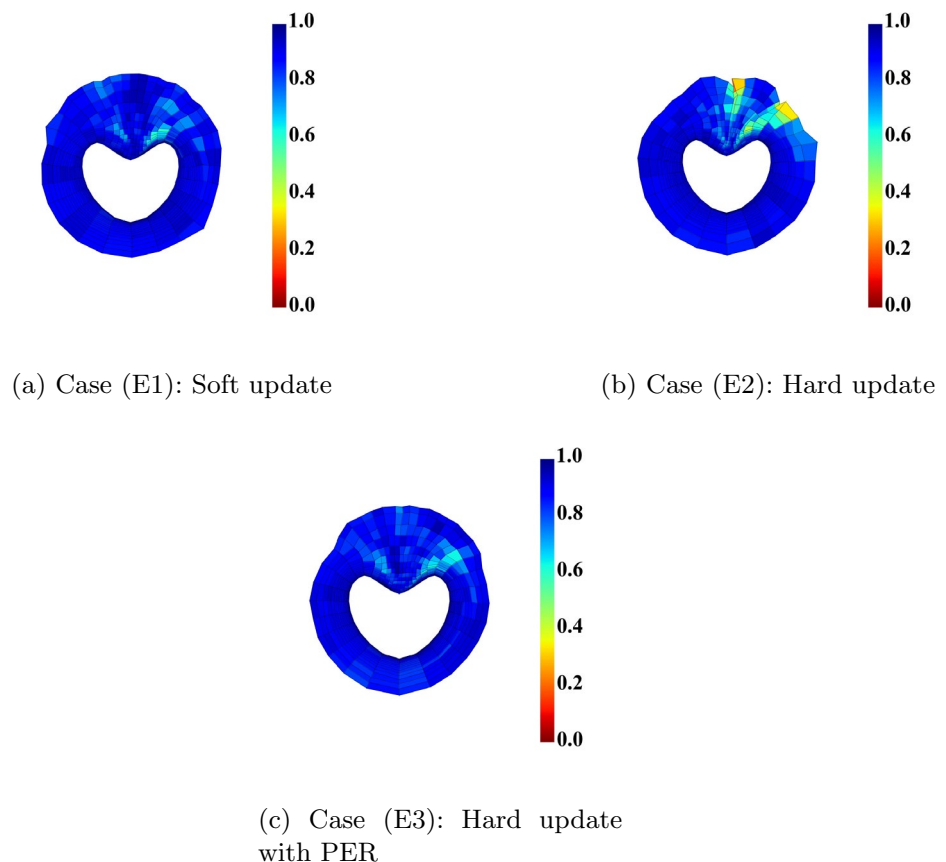
Figure 18 shows the grids generated using networks from the episodes with the highest rewards for each case. In case (D2), where learning failed, crossings occurred near the concave areas, resulting in low rewards. Conversely, the proposed learning strategy in case (D1) successfully built a grid of 20 layers at the 6,230-th episode, demonstrating better grid quality.

4.5 Study on target model update strategy

The DRL methodology used in this study employs the Double DQN technique, which includes two models: an online model and a target model. As described in Sect. 3.4, the target model does not directly undergo training, but periodically copies the weights from the online model. Depending on the method used to copy the weights, the update process can be categorized into hard and soft updates. The difference in learning outcomes between these two methods is explored in this analysis. Specifically, the PER technique can be employed to improve the learning efficiency in hard updates. Therefore, three scenarios were compared: soft update, hard update, and hard update with PER. Table 8 provides a description of each case. When a soft update is performed as expressed in Eq. (22), the weights of the online model must be gradually transferred to the target model, which requires a short update period. Conversely, with a hard update, as expressed in Eq. (21), the weights of the online model are transferred directly to the target model, which requires longer update periods for efficient learning. Therefore, the update frequency was set to one time step for the soft update and 10 episodes for the hard update.

Figure 19 depicts the learning progress for the cases listed in the table. As before, the y-axis represents the sum

Fig. 20 Quality of the grids generated by the network trained using varying model update strategies



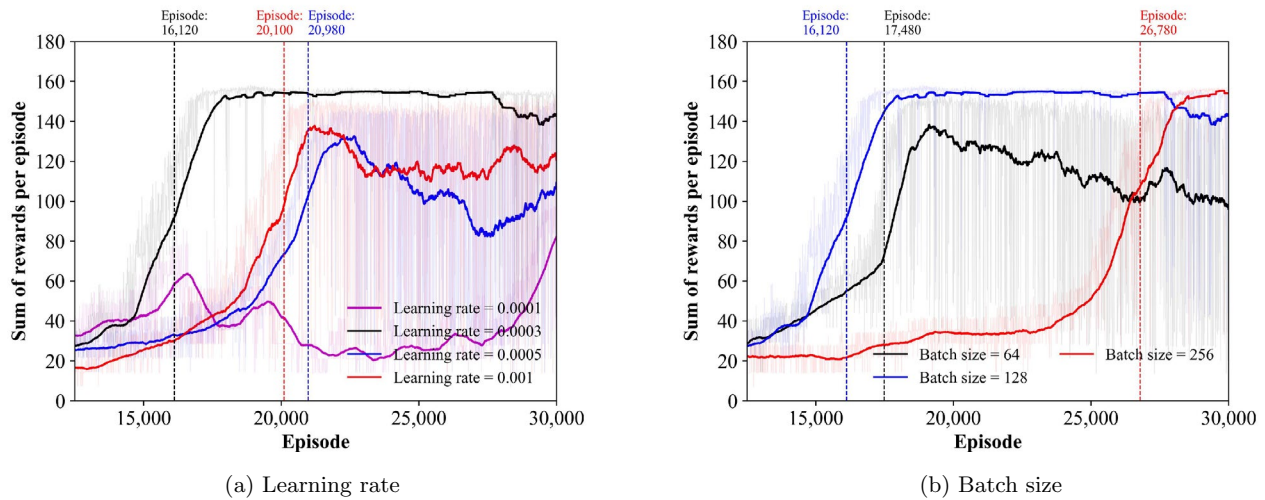


Fig. 21 Reward sums of meshes generated by the network trained using varying parameters

Table 9 Adjusting learning conditions according to an increase in the number of nodes

Parameter	Heart	Airfoil
Replay memory size	10,000	20,000
Batch size	128	256
Learning rate	0.001	0.0003

of the average rewards of the cells in each layer generated within an episode. The learning proceeded normally in all cases. However, the learning speed in the soft update case was the highest, achieving the highest average reward. In the case of the hard update, using the PER resulted in a reduction of approximately 2,000 episodes required to build a 20-layer grid. This confirms the effectiveness of using the

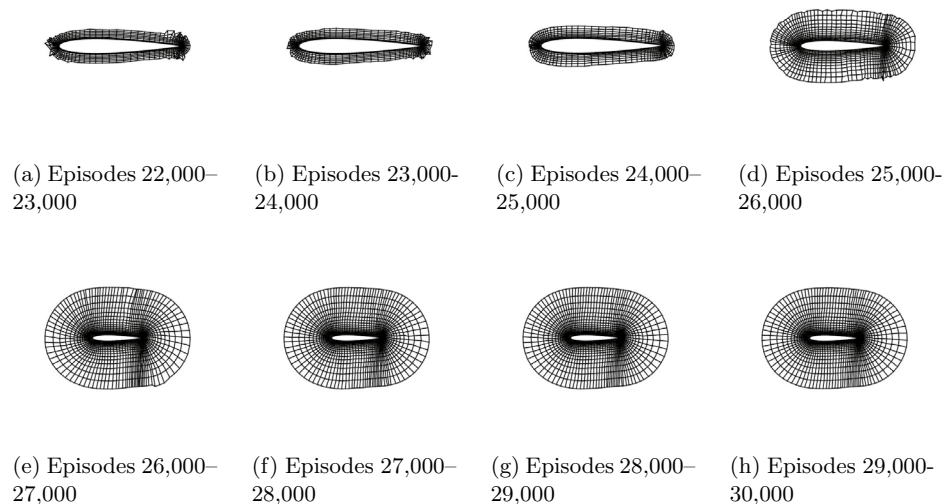
PER alongside hard updates. Despite this, the soft update without PER achieved the best training performance.

Figure 20 shows the grids generated using the network from the best-rewarded episode in each case. In all cases, the learning process successfully produced grids with 20 layers. However, in case (E2), low-quality cells were observed in the concave areas, resulting in a comparably lower grid quality.

4.6 Application to airfoil mesh generation

Based on the results from Sects. 3.1 to 3.6, the learning conditions for the mesh generation in Table 3 were derived and applied to airfoil grid generation. Under these conditions, learning progressed properly, and a successful grid

Fig. 22 Training process of the mesh generation for airfoil



of 20 layers was built by the 20,100-th episode (red line in Fig. 21a). Therefore, it was evident that the derived learning conditions were effective for airfoil geometry.

However, additional tuning of the learning parameters was required to refine the quality of the generated grid further. An additional case study was conducted to determine the learning rate. Figure 21a presents the results of the case study on the learning rate. While reducing the learning rate to 0.0005 from 0.001 resulted in no significant differences in the learning outcomes, a further reduction to 0.0003 resulted in notable improvements. At this rate, the point of stacking all layers was moved forward to approximately 4860 episodes, approaching sufficiently high rewards after training. Nevertheless, when it was reduced to 0.0001, learning nearly ceased, and progress was observed only around the 30,000-th episode.

One possible explanation for this is that, with more than three times the number of nodes used in training, the narrower spacing between nodes increased the likelihood of crossings, creating a more challenging environment in which actions must be more carefully tuned for effective learning.

Setting the learning rate to 0.0003 resulted in the generation of grids of satisfactory quality. However, reducing the learning rate excessively can significantly slow the learning progress.

Subsequently, a case study on the batch size was conducted. Although the shape of the airfoil consists only of convex curves, making it simpler than the heart shape, it poses different challenges owing to the presence of more nodes. To reflect these characteristics, adjustments to the batch size and the corresponding buffer size of the replay memory are necessary to train more node information simultaneously. Figure 21b depicts the results of the case study on the batch size. Reducing the batch size to half (64) significantly deteriorated the learning effectiveness. Doubling the batch size significantly delayed the learning process compared with using a batch size of 128, with the successful generation of 20 layers occurring 10,000 episodes later, although slightly higher maximum reward values were achieved. The ultimate goal of the grid generation tool is to produce high-quality grids through inference alone and not through training on individual shapes. Therefore, selecting a

Fig. 23 Quality of the grids according to the metrics

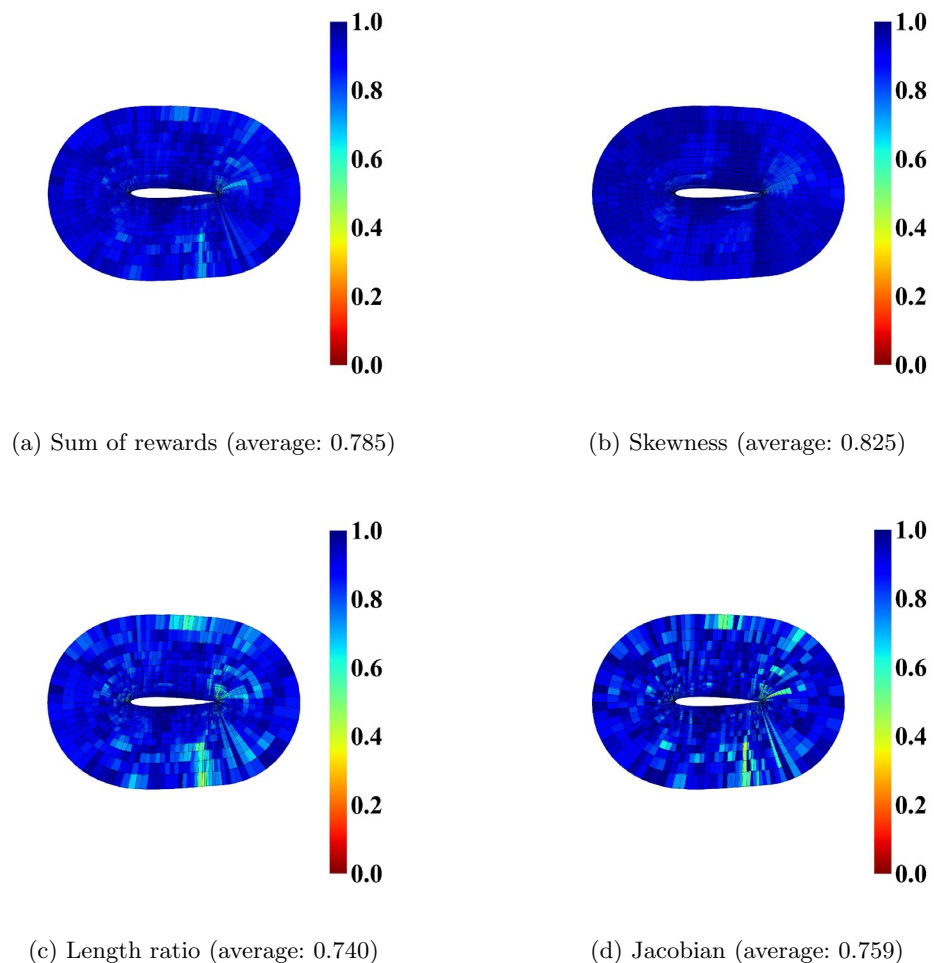
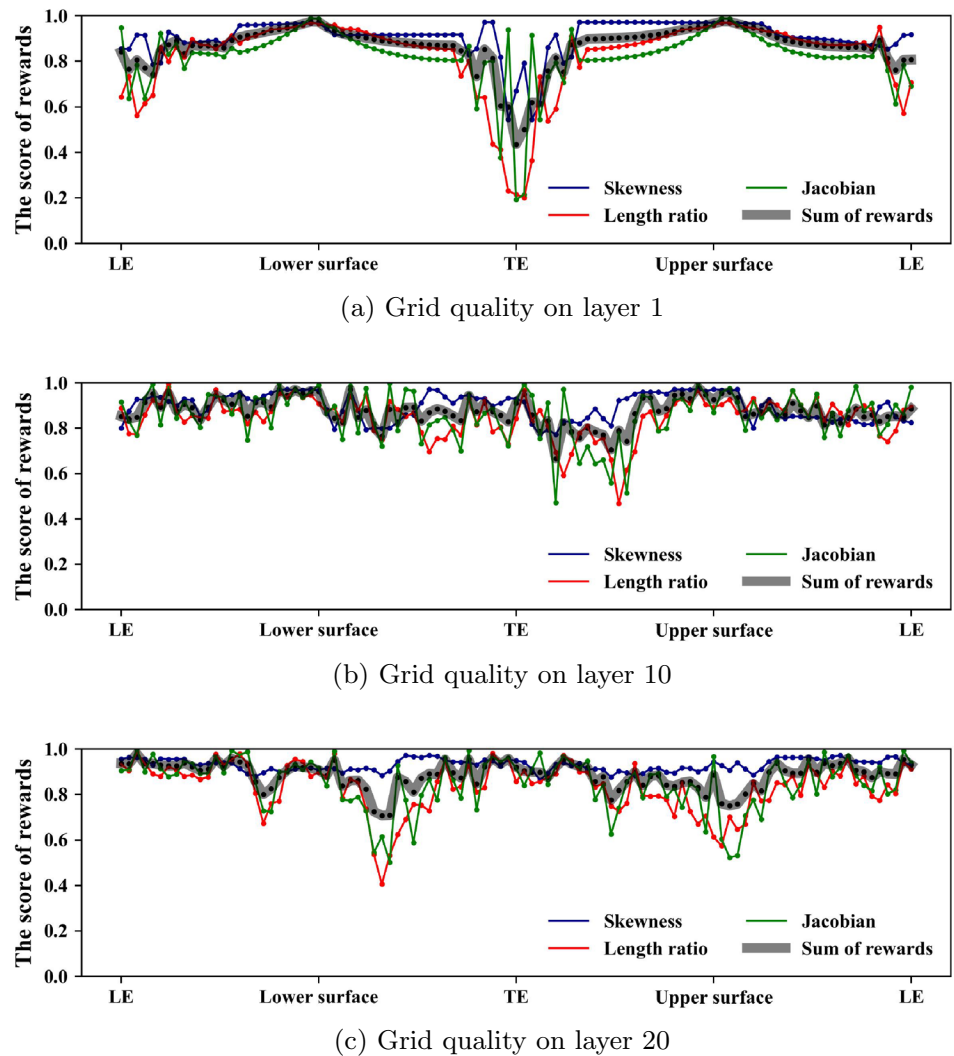


Fig. 24 Comparison of grid quality for each layer



model that yields better results is preferable, even if it incurs a slight loss in training time. Consequently, we opted for the model trained with a batch size of 256; the final learning conditions, including this, are provided in Table 9.

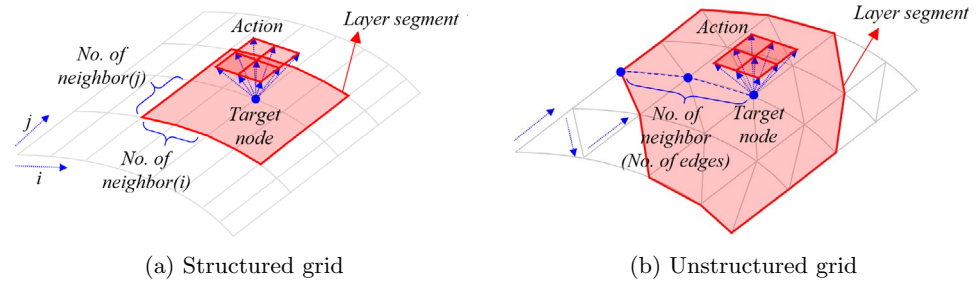
Figure 22 shows the evolution of the airfoil grid generated by the algorithm under the proposed conditions. Starting from episode 22,000, in which the effects of learning became apparent, to episode 30,000, grid generation was performed using networks from the episodes with maximum rewards at intervals of 1,000. Below episode 26,000, where a grid of 20 layers was not yet successfully stacked, most crossings occurred near the leading and trailing edges, where the nodes were densely clustered. In these areas, the range of actions available to the nodes was narrow and the learning involved exploring these options. After successfully stacking the 20-layer grid around episode 26,000, there was a phase of slight adjustments, leading to an enhancement in the grid quality. In Fig. 22h, the somewhat jagged appearance of

the outermost grid observed in Fig. 22e disappears, and the outer boundary becomes a smooth closed curve.

Figure 23 depicts the rewards of the generated grid and the numerical values of the mesh quality metrics that constitute the reward. The skewness showed the highest value among all metrics. This is because the skewness had the highest weight of 0.5. Nevertheless, all metrics showed satisfactory values of 0.7 or above.

Delving deeper into the distribution of rewards, the distribution of each reward metric for certain layers was examined. Figure 24a shows the distribution of reward metrics for the first layer, indicating a symmetric distribution similar to the previous layer (upper and lower surfaces), which was perfectly symmetrical. Near the leading and trailing edges, all metrics exhibited low values, whereas uniformly good rewards were obtained for both surfaces. (Since the term “surface” is generally used to describe the boundary of a two dimensional airfoil, this section uses “surface” instead of “boundary.”)

Fig. 25 Problem definition for three dimensional mesh generation



Examining the reward metric distribution at the 10th and 20th layers in Fig. 24b, c reveals the disappearance of symmetry between the upper and lower surfaces, with rewards increasing near the trailing and leading edges. Conversely, the rewards for the upper and lower surfaces tended to be lower, particularly in the 10-th layer, where the rewards on the upper surface near the trailing edge were below 0.5.

These observations indicate that the learning process promotes uniform and favorable rewards by growing the layers to ensure a certain level of consistency in the node-to-node distances. This is because obtaining a favorable reward in regions where the distances between nodes are relatively narrow is challenging.

In addition, because the Jacobian and length ratios are metrics that operate similarly, they exhibit similar trends. This similarity is further evident in Fig. 23, where the contours of Fig. 23c, d exhibit a resemblance.

4.7 Considerations for extension to three dimensions

The proposed agent-based mesh generator, based on DLR, was successfully developed in a two-dimensional example. Finally, a method for extending this technique to three dimensions is proposed to conclude the study.

The Fig. 25 illustrates the extension method to three dimensions for each grid type. In two dimensions, the state is a set of x and y coordinates on a curve. In three dimensions, it can be represented as a set of x , y , and z coordinates on a surface. For a structured grid, the state in the two-dimensional problem exists only in the i direction, but in three dimensions, as shown in Fig. 25a, it can be extended to the j direction, defining surface-like layer segments. An additional angular direction variable is added for the action. In an episode, a double inner loop, moving in the j direction while traversing in the i direction, is included within one time step. The reward uses metrics such as three-dimensional skewness and Jacobian, which are commonly used to evaluate three-dimensional mesh quality. When dealing with unstructured grids, the grid is considered as a type of graph. As shown in Fig. 25b,

Neighboring nodes are determined by the number of edges that need to be crossed to reach them. Actions and rewards can be determined in the same way as for structured grids. However, unlike structured grids, an episode does not require a double inner loop; a single traversal following the node numbers within one time step is sufficient.

Simply extending this methodology from two dimensions to three dimensions results in a significant increase in computational load. Therefore, appropriate efficiency and optimization strategies must be employed to address this.

5 Conclusions

This study was inspired by the idea that the boundary layer mesh generation problem can be formulated as an optimal pathfinding problem. To prove this, an agent-based mesh generation method was proposed by defining the boundary layer mesh generation as a DRL problem. The training conditions were thoroughly investigated and applied to the airfoil boundary layer mesh generation to train the proposed method. The results confirmed the viability of the proposed agent-based mesh generation approach. The key contributions of this study are as follows:

- (1) A new algorithm for generating grids using DRL is proposed. The proposed algorithm views the nodes of the surface grid as individual agents and solves the grid generation problem as an optimal pathfinding problem in which each agent determines the optimal path towards the outer space to create high-quality grids. To achieve this, the states (layer segments and layer count), actions (movement of nodes), and rewards (grid quality) were defined.
- (2) To train the grid generation problem as a DRL task, one of the main challenges, namely the non-deterministic rewards problem, was addressed by introducing a sub-episode loop. Sub-episodes, which are inner loops traversing all nodes within a layer in a single time step, were added to the main episode to address the incompleteness issue of actions that can be taken by a single agent (node).

- (3) Appropriate learning conditions have been derived. State members were proposed as layer segments and layer counts using a method suggested for normalizing the node positions of the layer segments. The inclusion of five neighboring nodes in each layer segment and a strategy for averaging the losses for nodes with the same index were also proposed to train the network. Finally, performing a soft update of the Double DQN target model at a period of one time step during training was recommended.
- (4) The derived learning conditions were applied to generate high-quality grids for airfoil, confirming the feasibility of creating quality grids while minimizing human intervention. This establishes a foundation for developing new grid algorithms that can effectively produce various grids with minimal human involvement.

Excluding human intervention and enabling grid generation for various shapes without the need for learning require further research. Additional studies utilizing datasets with diverse shapes for training will be addressed in the future work.

Acknowledgements This work was supported by Korea Institute of Industrial Technology (KITECH JH-24-0003) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2022R1F1A1073100).

Author contributions K.Lee and S.Lee. wrote the main manuscript text and K.Lim. conducted experiments and prepared figures. K.Yee reviewed the manuscript.

Data availability No datasets were generated or analysed during the current study.

Declarations

Conflict of interest The authors declare no competing interests.

Open Access This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

References

1. Rizzi A, Luckring JM (2021) Historical development and use of cfd for separated flow simulations relevant to military aircraft. *Aerosp Sci Technol* 117:106940
2. Raveh DE (2007) Cfd-based models of aerodynamic gust response. *J Aircr* 44(3):888–897
3. Yang S, Yee K (2022) Design rule extraction using multi-fidelity surrogate model for unmanned combat aerial vehicles. *J Aircr* 59(4):977–991
4. Spalart PR, Venkatakrishnan V (2016) On the role and challenges of cfd in the aerospace industry. *Aeronaut J* 120(1223):209–232
5. Ba D-C, Deng W-J, Che S-G, Li Y, Guo H-X, Li N, Yue X-J (2016) Gas dynamics analysis of a rotary compressor based on cfd. *Appl Therm Eng* 99:1263–1269
6. Xu K, Liu C (2017) A paradigm for modeling and computation of gas dynamics. *Phys Fluids* 29(2):026101
7. Sande P, Ray S (2014) Mesh size effect on cfd simulation of gas-fluidized geldart a particles. *Powder Technol* 264:43–53
8. Xie ZQ, Sevilla R, Hassan O, Morgan K (2013) The generation of arbitrary order curved meshes for 3d finite element analysis. *Comput Mech* 51:361–374
9. Katz A, Sankaran V (2011) Mesh quality effects on the accuracy of cfd solutions on unstructured meshes. *J Comput Phys* 230(20):7670–7686
10. Bern MW, Plassmann PE (2000) Mesh generation. In: *Handbook of computational geometry*, vol 38
11. Sørensen NN (1998) Hypgrid2d. a 2-d mesh generator
12. Thompson JF, Soni BK, Weatherill NP (1998) *Handbook of Grid Generation*. CRC Press, Boca Raton
13. Thomas P, Middlecoff J (1980) Direct control of the grid point distribution in meshes generated by elliptic equations. *AIAA J* 18(6):652–656
14. Steger J, Sorenson R (1979) Automatic mesh-point clustering near a boundary in grid generation with elliptic partial differential equations. *J Comput Phys* 33:405–410
15. Bertsekas D (2019) *Reinforcement learning and optimal control*. Athena Scientific, Nashua
16. Henderson P, Islam R, Bachman P, Pineau J, Precup D, Meger D (2018) Deep reinforcement learning that matters. In: *Proceedings of the AAAI conference on artificial intelligence*, vol 32
17. Cervenka J, Wessner W, Al-Ani E, Grasser T, Selberherr S (2006) Generation of unstructured meshes for process and device simulation by means of partial differential equations. *IEEE Trans Comput Aided Des Integr Circuits Syst* 25(10):2118–2128
18. Kiran BR, Sobh I, Talpaert V, Mannion P, Al Sallab AA, Yogamani S, Pérez P (2021) Deep reinforcement learning for autonomous driving: a survey. *IEEE Trans Intell Transp Syst* 23(6):4909–4926
19. Kalapos A, Górcs, Moni R, Harmati I (2020) Sim-to-real reinforcement learning applied to end-to-end vehicle control. In: *2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR)*. IEEE, pp 1–6
20. Lee K, Kim S, Lim S, Choi S, Hong M, Kim JI, Park Y-L, Oh S (2020) Generalized tsallis entropy reinforcement learning and its application to soft mobile robots. In: *Robotics: science and systems*, vol 16, pp 1–10
21. Zhang Z, Wang Y, Jimack PK, Wang H (2020) Meshingnet: a new mesh generation method based on deep learning. In: *International conference on computational science*. Springer, pp 186–198
22. Pan J, Huang J, Cheng G, Zeng Y (2023) Reinforcement learning for automatic quadrilateral mesh generation: a soft actor-critic approach. *Neural Netw* 157:288–304
23. Kim I, Kim S, You D (2024) Non-iterative generation of an optimal mesh for a blade passage using deep reinforcement learning. *Comput Phys Commun* 294:108962

24. Kaelbling LP, Littman ML, Moore AW (1996) Reinforcement learning: a survey. *J Artif Intell Res* 4:237–285
25. Watkins CJ, Dayan P (1992) Q-learning. *Mach Learn* 8:279–292
26. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G (2015) Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533
27. Rodrigues Gomes E, Kowalczyk R (2009) Dynamic analysis of multiagent q-learning with ϵ -greedy exploration. In: *Proceedings of the 26th annual international conference on machine learning*, pp 369–376
28. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: *Proceedings of the AAAI conference on artificial intelligence*, vol 30
29. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In: *International conference on machine learning*. PMLR, pp 1928–1937
30. Schaul T, Quan J, Antonoglou I, Silver D (2015) Prioritized experience replay. *arXiv preprint [arXiv:1511.05952](https://arxiv.org/abs/1511.05952)*
31. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser Ł, Polosukhin I (2017) Attention is all you need. In: *Advances in neural information processing systems*, vol 30
32. Chen X, Gong C, Liu J, Pang Y, Deng L, Chi L, Li K (2022) A novel neural network approach for airfoil mesh quality evaluation. *J Parallel Distrib Comput* 164:123–132
33. Pointwise (2005) *Gridgen user's manual*. Pointwise Inc, Fort Worth

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Authors and Affiliations

Keunoh Lim¹ · Kyungjae Lee² · Sanga Lee³ · Kwanjung Yee¹

✉ Sanga Lee
ivory@kitech.re.kr

✉ Kwanjung Yee
kjyee@snu.ac.kr

Keunoh Lim
keenok@snu.ac.kr

Kyungjae Lee
kyungjae.lee@ai.cau.ac.kr

¹ Department of Aerospace Engineering, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 08826, Republic of Korea

² Department of Artificial Intelligence, Chung-Ang University, 84 Heukseok-ro 5-gil, Dongjak-gu, Seoul 06910, Republic of Korea

³ Manufacturing AI Research Center, Korea Institute of Industrial Technology, 156 Gaetbeol-ro, Yeonsu-gu, Incheon 21999, Republic of Korea