Optimal Experimental Design using Reinforcement Learning - Final Report

Apoorv Srivastava [*1] **Khoa Tran** [*1] **Mehdi Zouiten** [*1]

Abstract

Optimal experimental design (OED) is essential for parameter estimation in spatiotemporal models, particularly when data collection is expensive or noisy. However, the combinatorial complexity of OED renders traditional optimization methods computationally infeasible in high-dimensional settings. Reinforcement learning (RL) has emerged as a promising approach for tackling such optimization problems, with prior work demonstrating its effectiveness in OED using Temporal Difference (TD) learning. In this study, we extend this work by employing Deep Q-Learning (DQN) with experience replay and fixed Q-targets to improve scalability and generalization in high-dimensional settings. Additionally, we explore the OED problem using Monte Carlo Tree Search (MCTS), framing it as a single-player game. The code for our implementation can be found at: https://github.com/apoorv-s/Optimal-Experimental-Design-using-RL

1. Introduction

Optimal experimental design (OED) is a fundamental challenge in scientific research, engineering, and data-driven decision-making. It aims to maximize the information gained from experiments while minimizing costs, uncertainties, and resource consumption. Model-based OED techniques rely on exploiting the observability Gramian or the observability matrix (Singh & Hahn, 2005), which involve estimating observability covariance matrices of systems and quickly become challenging even for moderately complex dynamic systems.

With m sensors and n possible sensor locations, the OED problem is NP-hard (Krause et al., 2008) with combinatorial complexity. Traditional approaches to OED often rely on greedy methods or heuristics-based optimization techniques, such as simulated annealing and genetic algorithms (Yang et al., 2019). However, these methods can be computationally expensive and may struggle with complex dynamic systems even with moderate dimensions.

Reinforcement learning (RL) has emerged as a promising paradigm for adaptive decision-making in sequential environments and has demonstrated strong capabilities in addressing combinatorial optimization problems (Mazyavkina et al., 2020). RL has been used to study the OED problem by formulating it as a sequential decision-making problem (Wang et al., 2020). The prior work over OED in distributed parameter systems (DPS) by (Wang et al., 2020) looks at temporal difference (TD)-based method to identify the optimal sensor location that best reconstructs the original DPS systems.

In this work, we build on the prior work using Deep Q-Learning (DQN) to extend the methodology proposed in (Wang et al., 2020) for generalization to continuous action spaces and scalability to higher dimensions. We plan to examine the DQN-based OED on a wide variety of two-dimensional systems. The first two examples look at hyperbolic systems in the form of an advection equation and an inviscid Burgers equation, where the information travels at a finite speed, and the third example looks at a parabolic system in the form of an advection-diffusion-reaction equation where information travels at an infinite speed.

We also examine the OED problem using the Monte Carlo Tree Search (MCTS) algorithm used to train AlphaZero (Silver, 2018). The OED problem is framed as a single-player game, and MCTS is used to identify the best action in any given state. The results for the MCTS are limited to the advection system only.

2. Methodology

We consider the OED problem for a general class of DPS,

$$\frac{\partial u(\mathbf{x},t)}{\partial t} = \mathcal{L}\left(u, \nabla u, \nabla^2 u, \ldots, \nabla^n u\right) + F(u, \mathbf{x}, t),$$

describing evolution of the state $u(\mathbf{x},t) : \Omega \times \mathbb{R}^+ \to \mathbb{R}$ over the domain $\mathbf{x} \in \Omega \subset \mathbb{R}^2$ subject to the initial condition $u(\mathbf{x},0) = u_0(\mathbf{x})$ and arbitrary boundary conditions. To

[*]Equal contribution [1]Stanford University. Correspondence to: Apoorv Srivastava <apoorv1@stanford.edu>, Khoa Tran <khoa98@stanford.edu>, Mehdi Zouiten <zouiten@stanford.edu>.

Final project of CS234 Stanford Winter 2025, CA, USA, 2025.

mitigate the challenges associated with OED for infinite-dimensional $u(\mathbf{x}, t)$, we use Karhunen-Loève decomposition (KLD) to approximate the state $u(\mathbf{x}, t)$ using a set of orthogonal spatial basis functions (BFs). This approach reduces the system's dimensionality while retaining its dominant dynamics, enabling efficient OED for accurate state estimation.

Generally, the state $u(\mathbf{x}, t)$ is computed numerically on a discrete spatial grid over discrete time steps. Representing the solution as $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_K]$, where each column $\mathbf{y}_k$ corresponds to the state $u(\mathbf{x}_{1:n}, t_k)$ at time t_k over an n-point spatial grid.

In discrete settings, KLD is equivalent to Principal Component Analysis (PCA), as both methods involve eigenvalue decomposition of the covariance matrix (Gerbrands, 1981). Following (Wang et al., 2020), the covariance matrix $\mathbf{R} \in \mathbb{R}^{n \times n}$, which captures the spatial correlation between different locations over time, is computed using columns of $\mathbf{Y}$. The corresponding eigenvalue value problem $\mathbf{R}\mathbf{\Phi} = \mathbf{\Lambda}\mathbf{\Phi}$ results in matrix $\mathbf{\Phi}$ with eigenvectors $\mathbf{\Phi}_i$ (KLD modes) and corresponding eigenvalues λ_i in the diagonal matrix $\mathbf{\Lambda}$. The eigenvalues represent the variance captured by the corresponding modes and are sorted in descending order, ensuring the most significant modes are prioritized. The total energy captured by the first k modes is

$$E_k = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{j=1}^{n} \lambda_j}.$$

We choose k such that $E_k \geq 0.99$, guaranteeing that the selected modes retain 99% of the system's variance, significantly reducing complexity while maintaining essential dynamics (Wang et al., 2020).

Once the dominant modes are selected, the discretized system state can be approximated using only the first L KLD modes,

$$\hat{\mathbf{y}}_k = \sum_{i=1}^{L} \boldsymbol{\phi}_i c_i(t_k),$$

such that $\mathbf{y}_k = \hat{\mathbf{y}}_k + \epsilon$, where $c_i(t) = \langle \boldsymbol{\phi}_i, u(\mathbf{x}, t) \rangle$ and ϵ represents the noise in the approximation due to limited number of KLD modes.

Following (Wang et al., 2020), using m sensor locations, defined via a permutation matrix $\mathbf{P}_m \in \mathbb{R}^{m \times n}$, the measurements $\mathbf{y}^m \in \mathbb{R}^m$ for any arbitrary k can be represented as

$$\mathbf{y}^m = \mathbf{P}_m \mathbf{y} = \mathbf{P}_m \mathbf{\Phi} \mathbf{c} + \mathbf{P}_m \epsilon, \tag{1}$$

where $\mathbf{\Phi}$ is a matrix representation of the KLD model, $\mathbf{c}$ is the vector of corresponding coefficients, and $\mathbf{P}_m \epsilon$ is the projection error.

The OED problem focuses on identifying the best sensor location, i.e., $\mathbf{P}_m$ that minimizes the sum of squared projection error.

2.1. RL Formulation

The OED problem is formulated as a Markov Decision Process (MDP) to effectively leverage the RL framework. We consider a spatial domain represented via an $n_x \times n_y$ discretized regular rectangular grid with a total of $n = n_x n_y$ points.

2.1.1. State Representation

The dimensional state s represents the current placement of m sensors on the spatial grid and is represented via the permutation matrix $\mathbf{P}_m \in \mathbb{R}^{m \times n}$, in which each row has a single element, corresponding to the i-th sensor location, set to 1 and the remaining elements are set to 0. In the implementation, we also maintain an equivalent representation of the state as a tuple of m sensor locations on the grid.

2.1.2. Action Spaces

We consider two action space formulations that model the movement of sensors on the spatial grid and compare their impact on learning efficiency and performance across examples in section 3.

The first action space, in line with (Wang et al., 2020), formulates an action as selecting one of the m sensors and moving to an empty location on the grid. With m sensors and $(n - m)$ empty locations, the action space size is $|\mathcal{A}| = m(n - m)$. The action space is represented via a discrete space and can take values $0, 1, 2, \ldots, m(n - m) - 1$. Each value can then be mapped to a specific sensor and an empty location on the grid. This allows any sensor to move to any empty location in a single step, enabling large spatial jumps. We refer to this action space as the old action space, and it is represented by $\mathcal{A}_{\text{old}}$.

Alternatively, we define the new action space $\mathcal{A}_{\text{new}}$ under which a sensor is allowed to move only to an adjacent empty grid cell in a single step, constraining sensor movement to local grid points. This significantly reduces the size of the action space since, for each sensor, there are only four possible movements (up, down, left, right), such that $|\mathcal{A}_{\text{new}}| = 4m$. If the movement of the sensor in the chosen direction is not possible (due to domain boundaries or occupied locations), the action has no effect. This local movement constraint enforces spatial continuity and may improve exploration of the state space by preventing large jumps.

2.1.3. Reward Function

The reward function focuses on minimizing the sum of squared projection error defined in Eq. 1. Assuming that

the noise in the measurements ϵ is i.i.d. Gaussian with zero mean and fixed variance, the problem of simplifying the projection error can be simplified to that of maximizing the $\det(\mathbf{T}_m)$, where $\mathbf{T}_m = \mathbf{\Phi}^T \mathbf{P}_m^T \mathbf{P}_m \mathbf{\Phi}$.

The reward function can then be directly defined as

$$r(s_t) = \det(\mathbf{T}_m) = \det(\mathbf{\Phi}^T \mathbf{P}_m^T \mathbf{P}_m \mathbf{\Phi}),$$

which in turn minimizes the reconstruction error from the observation for any given state s_t defined via $\mathbf{P}_m$.

Algorithm 1 KLD and Reward Computation

Require: Solution data $\mathbf{Y}$, energy threshold $\varepsilon = 0.99$
$\bar{\mathbf{Y}} \leftarrow \frac{1}{m} \sum_{j=1}^{m} \mathbf{Y}[:, j]$
$\mathbf{Y}_c \leftarrow \mathbf{Y} - \bar{\mathbf{Y}}$
$\mathbf{R} \leftarrow \frac{1}{m-1} \mathbf{Y}_c \mathbf{Y}_c^T$
$\{\lambda_i, \phi_i\}_{i=1}^{n} \leftarrow \text{EigenDecomposition}(\mathbf{R})$
$r \leftarrow \min\{j : \frac{\sum_{i=1}^{j} \lambda_i}{\sum_{i=1}^{n} \lambda_i} \geq \varepsilon\}$
$\mathbf{\Phi} \leftarrow [\phi_1, \ldots, \phi_r]$
for sensor locations $\mathbf{P}_m$ **do**
    $\mathbf{T}_m \leftarrow \mathbf{\Phi}^T \mathbf{P}_m^T \mathbf{P}_m \mathbf{\Phi}$
    $Reward \leftarrow \det(\mathbf{T}_m)$
end for

2.1.4. Episode Termination

An episode terminates when the maximum number of steps $t_{\max}$ is reached. This early termination strategy prevents the agent from exploring non-improving trajectories and enhances learning efficiency. For $\mathcal{A}_{\text{old}}$, smaller values of $t_{\max}$ (around 5-10) are used since optimal configurations can be reached quickly with large jumps. For $\mathcal{A}_{\text{new}}$, much larger values (around 500) are used to accommodate the local movement constraints.

2.2. DQN Implementation

The OED problem environment is set up to allow for using a Deep Q-Network (DQN) through the `stable-baselines3` package (Raffin et al., 2021). The implementation includes standard DQN enhancements such as experience replay to break correlations between consecutive samples, a target network with periodic updates to stabilize learning, ϵ-greedy exploration strategy with an annealing schedule, and Double DQN to reduce overestimation bias.

To mitigate numerical instabilities, the eigendecomposition of the covariance matrix and KLD modes are pre-computed before training begins. This ensures consistency in the reward function across episodes and reduces computational overhead during the RL training process.

The rewards computation is summarized in Algorithm 1, and the DQN-based OED is carried out using Algorithm 2 via `stable-baselines3` package (Raffin et al., 2021).

Algorithm 2 Optimal Sensor Placement with DQN

Initialize the environment with observation space as a 2D grid containing binary values depicting the sensor placement
Define action $a \in \mathcal{A}(s)$, where for each s, there are $m(n-m)$ available actions of moving one of the sensors to one of the new empty positions
Call the PDE solver to retrieve dynamics $\mathbf{Y}$
Initialize the recorded maximum episode reward $\mathcal{F}$: $\mathcal{F}_{\max} \leftarrow 0$
Set maximum time steps in each episode: $t_{\max}$
while $t \leq t_{\max}$ **do**
    Initialize $t \leftarrow 1$, s_t
    Select action a_t using DQN policy
    Take a_t, observe next-state placement $\mathcal{P}_m$ and reward $\mathcal{F}_m$
    Store $\mathcal{P}_m$ and $\mathcal{F}_m$ in the learning buffer and update DQN policy
    if $\mathcal{F}(s_t) > \mathcal{F}_{\max}$ **then**
        $\mathcal{F}_{\max} \leftarrow \mathcal{F}(s_t)$, $s^* \leftarrow s_t$
    end if
end while
Obtain optimal sensor locations: $\mathcal{P}_m^* \leftarrow s^*$

2.3. Monte Carlo Tree Search (MCTS)

Inspired by AlphaZero (Silver, 2018), we adapt its framework to the OED problem by formulating it as a single-player game. Prior to selecting an action, the agent performs a tree search based on the Monte Carlo (MC) method through the end of the episode without altering the state. The agent then chooses the most-visited action/state from the tree search as its next move. This setup enables the agent to plan ahead and make optimal decisions.

The MCTS framework consists of a search tree and a two-headed value-policy neural network that work iteratively to refine sensor positions. At each step, the neural network guides the expansion of the tree, and the most visited action is selected as the next action. The resulting trajectories and value estimates are then used to further train the network, where the policy head learns to match improved action probabilities from MCTS, and the value head is optimized to predict actual returns.

2.3.1. Search tree

During the search step, the root of the tree is set to the current state of the environment, and the tree construction comprises two stages: expansion and backpropagation. The critical hyperparameters that govern the search tree expansion include `max_node`, which defines the total number of visited nodes (including duplicates) before selecting the

best action, and `max_depth`, which represents the maximum depth of any branch. The `max_node` is determined by the computational budget available, and the `max_depth` is set to the remaining steps in the episode to ensure consistency between the neural network's predictions and value backpropagation.

The tree expansion begins at the root, where any node n has total visits $N(n)$, state $s(n)$, state value $V(n)$, and the reward $R(n)$ from its parent n_p as attributes. At each node, the neural network predicts $V(n)$ and the prior probabilities $P(n_c)$ for the children nodes. The best child node (and action) is selected using the Upper Confidence Bound (UCB) score,

$$Q\_score(n_c) = R(n_c) + \gamma * V(n_c)$$

$$Prior\_score(n_c) = prior\_scale * P(n_c) * \frac{\sqrt{N(n)}}{N(n_c)+1}$$

$$UCB\_score(n_c) = Q\_score(n_c) + Prior\_score(n_c)$$

$$a(n_c) = argmax(UCB\_score(n_c)),$$

where γ is the discount factor, and the $prior\_scale$ controls exploration of unvisited nodes. For unvisited nodes, $V(n_c)$ = 0, effectively masking the Q_score. This follows the bandit approach to balance exploration and exploitation. Expansion continues until an unvisited node or max depth is reached.

At an unvisited node or at maximum depth, backpropagation updates the values along the branch up to the root node. At each node n,

$$N(n) \leftarrow N(n) + 1$$

$$G(n) \leftarrow R(n) + \gamma * V(n)$$

$$V(n_p) \leftarrow V(n_p) + \frac{1}{N(n_p)+1} * (G(n) - V(n_p))$$

$$n \leftarrow n_p$$

. This resembles an incremental Monte Carlo update. Once the root is reached, another expansion-backpropagation cycle begins until `max_node` is reached. The tree then selects the best action as the most-visited child node of the root and returns the action priors based on the visit frequencies.

2.3.2. Value-Policy Neural Network

The Value-Policy neural network is a two-headed architecture that predicts the state value and action probabilities given the current state. In this implementation, we use a simple multilayer perceptron with fully connected hidden layers and ReLU activations. The value head consists of a linear layer, while the policy head includes an additional softmax layer to produce action probabilities.

After MCTS selects the best action, a training tuple (s_t, π_t, V_t) is stored in memory, where s_t is the environment state, π_t represents the action probability targets, and V_t is the value estimate. The value target is computed as the discounted sum of future rewards from state s_t,

$$V(s_t) = \sum_{j=t}^{T} \gamma^{j-t} R(s_j).$$

Once the episode concludes, these tuples are processed and added to the training dataset.

Training is performed in batches at the end of each episode using the Adam optimizer. The loss function is the sum of cross-entropy loss for the policy head and mean squared error loss for the value head.

3. Results

The results are presented for three two-dimensional DPS and include the advection equation, the inviscid Burgers equation, and the advection-diffusion-reaction system. The advection equation and the Burgers equation are hyperbolic systems that model wave-like phenomena in which the information travels at a finite speed. In the advection system, the wave travels with a constant speed and maintains a smooth solution, and therefore, KLD modes, throughout. In the inviscid Burgers equation, shocks develop over time due to solution-dependent local wave velocity, leading to discontinuities in the solution and sharp gradients in the KLD modes. The solutions for the hyperbolic systems are obtained using `Pyclaw` solvers (Mandli et al., 2016). The advection-diffusion-reaction is a parabolic system in which the information travels at infinite speed, but the information attenuates as we move away from the source.

For each example, we study the OED problem using DQN under the old action space $\mathcal{A}_{\text{old}}$ and the new action space $\mathcal{A}_{\text{new}}$ and are benchmarked against results obtained using genetic algorithms (GA). The discount factor $\gamma = 1$ for all cases since we define the setup as a finite horizon problem. We consider a significantly longer horizon under $\mathcal{A}_{\text{new}}$ compared to $\mathcal{A}_{\text{old}}$ since under $\mathcal{A}_{\text{new}}$, a single sensor is allowed to move only one position at a time, and as a result, it may require many more steps to reach the optimal positions. Under $\mathcal{A}_{\text{old}}$, a sensor can move to any arbitrary position in the domain in a single step. The results for the OED problem using MCTS under $\mathcal{A}_{\text{new}}$ are limited to the advection equation system.

3.1. Example 1: Advection equation

The first example looks at the 2D advection equation,

$$\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u = 0, \tag{2}$$

which describes the transport of a scalar quantity $u(\mathbf{x}, t) : \Omega \times \mathbb{R}^+ \to \mathbb{R}$ in a fluid flow with the velocity field $\mathbf{v} =$

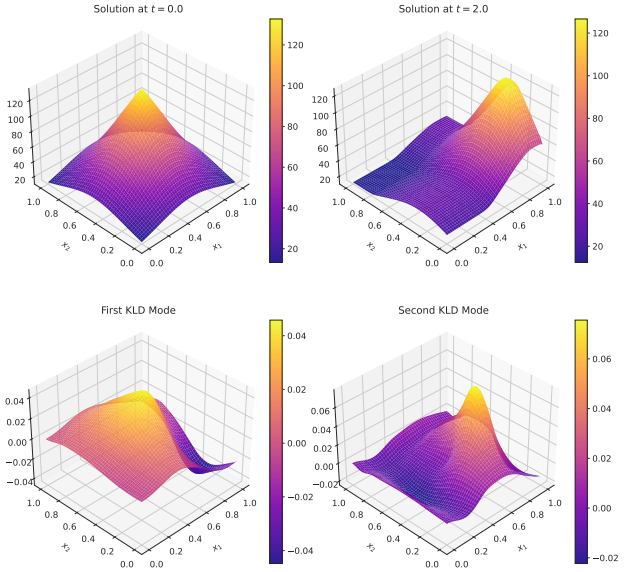

Figure 1. Temporal evolution of the state $u(\mathbf{x}, t)$ for advection equation and the corresponding first two KLD modes.

(v_x, v_y). It is a hyperbolic system where the information travels with finite speed, and the solution to Eq. 2 is obtained under extrapolated boundary conditions. The results are presented for $\Omega = [0, 1] \times [0, 1] \subset \mathbb{R}^2$ discretized using 50×50 regular grid and $t \in [0, 2]$ with Gaussian-like initial condition

$$u(\mathbf{x}, 0) = \frac{1.0}{1.0 + \exp(-0.75 + 5.0\|\mathbf{x} - 0.5\|)}.$$

The solution to Eq. 2 and the corresponding first two KLD modes are shown in Fig. 1.

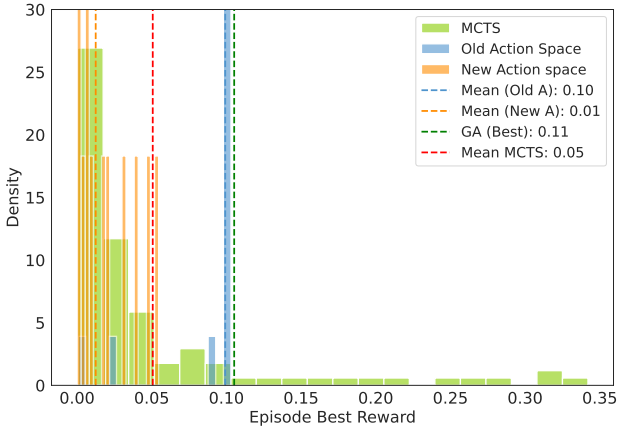

Figure 2. Distribution of reward using DQN with $\mathcal{A}_{old}$ and $\mathcal{A}_{new}$ and the results from GA for the advection equation system.

Under the 50×50 grid, we get $n = 2500$, and the OED considers the problem of placing $m = 2$ sensors over the domain to minimize the reconstruction error by maximizing $\det(T_m)$. We use DQN under $\mathcal{A}_{\text{old}}$ with $H = 5$ and under $\mathcal{A}_{\text{new}}$ with $H = 500$. The distribution of the rewards obtained using DQN with $\mathcal{A}_{old}$ and $\mathcal{A}_{new}$ and the maximum award obtained using GA is shown in Fig. 2.

The results suggest that the DQN with $\mathcal{A}_{\text{old}}$ gets close to the maximum reward obtained using GA most of the time, whereas the distribution of rewards obtained under $\mathcal{A}_{\text{new}}$ is concentrated on smaller values indicating instability in the DQN training. This is consistent with the episode rewards and loss values observed during training. For the MCTS, we are able to achieve much higher rewards in some cases compared to other algorithms, including GA. However, for most runs, the rewards are concentrated around zero. This could be due to the limited computational budget for MCTS and/or limited horizon length and requires further investigation.

3.2. Example 2: Inviscid Burgers Equation

The second example considers the 2D inviscid Burgers' equation,

$$\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u + u \nabla \cdot \mathbf{v} = 0, \tag{3}$$

where $u(\mathbf{x}, t) : \Omega \times \mathbb{R}^+ \to \mathbb{R}$ represents the scalar

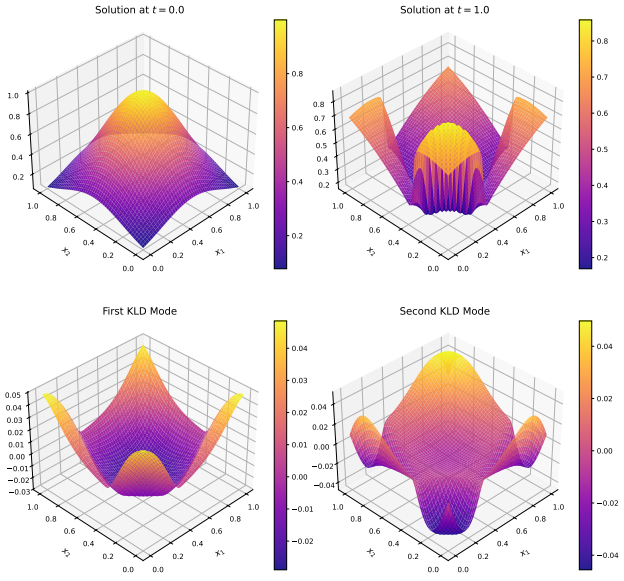

Figure 3. Shock formation in the solution of the 2D inviscid Burgers' equation.

field (such as velocity or density) over the domain $\Omega = [0, 1] \times [0, 1] \subset \mathbb{R}^2$ with periodic boundaries with a bivariate Gaussian initial condition center at $\mathbf{x}_0 = [0.5, 0.5]$ with $\Sigma = 0.3\mathbf{I}$. The inviscid Burgers' equation is also a hyperbolic PDE that exhibits shock formation and wave steepening, leading to discontinuities in the solution over

time. The evolution of $u(\mathbf{x}, t)$ over $t \in [0, 1]$ is examined, and the temporal snapshots of the solution and the dominant KLD modes are shown in Fig. 3.

Using $H = 10$ under $\mathcal{A}_{\text{old}}$ and $H = 500$ under $\mathcal{A}_{\text{new}}$, the distribution of the rewards obtained using DQN with $\mathcal{A}_{old}$ and $\mathcal{A}_{new}$is shown in Fig. 4. The maximum reward using GA was found to be 0.0764.

In this case, the DQN under $\mathcal{A}_{\text{new}}$ performs marginally better than the DQN under $\mathcal{A}_{\text{old}}$. However, DQN results are much smaller than that obtained using GA. The training loss decays, but the rewards fluctuate around low values, suggesting that the system gets stuck in local minima and is not able to get out. This behavior is consistent across multiple runs under varying network architectures. The challenges could be due to discontinuities in the solution and sharp gradients in the KLD modes that make the OED problem significantly harder and prevent generalization using DQN.

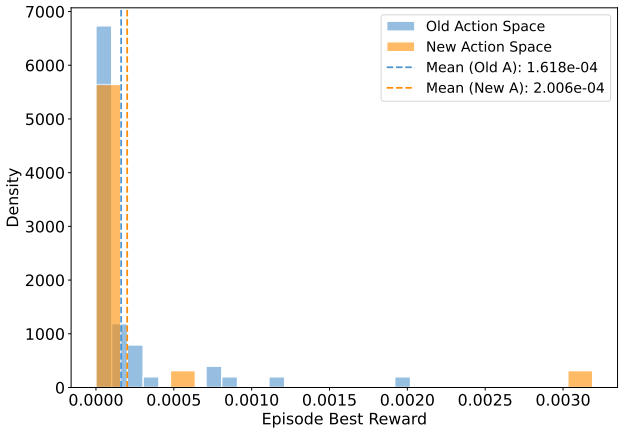

Figure 4. Distribution of reward using DQN with $\mathcal{A}_{old}$ and $\mathcal{A}_{new}$ and the results from GA for the inviscid Burgers system.

3.3. Example 3: Advection-Diffusion-Reaction system

This example examines the Advection-Diffusion-Reaction (ADR) system,

$$\frac{\partial u}{\partial t} + \mathbf{v} \cdot \nabla u = D\nabla^2 u + R(u, \mathbf{x}, t), \tag{4}$$

where $u(\mathbf{x}, t) : \Omega \times \mathbb{R}^+ \to \mathbb{R}$ with $\Omega = [0, 1] \times [0, 1] \subset \mathbb{R}^2$ is the scalar field (such as concentration, temperature, or pollutant level), and $\mathbf{v} \in \mathbb{R}^2$ is the velocity field, D is the diffusion coefficient, and $R(u, \mathbf{x}, t)$ is the reaction term. The ADR system is a parabolic system where the information travels at an infinite speed but with decaying influence. The evolution of the state over $t \in [0, 3]$ and the first two KLD modes are presented in Fig. 4

The results for this case are similar to that of the inviscid Burgers system. The mean reward from DQN under $\mathcal{A}_{\text{new}}$

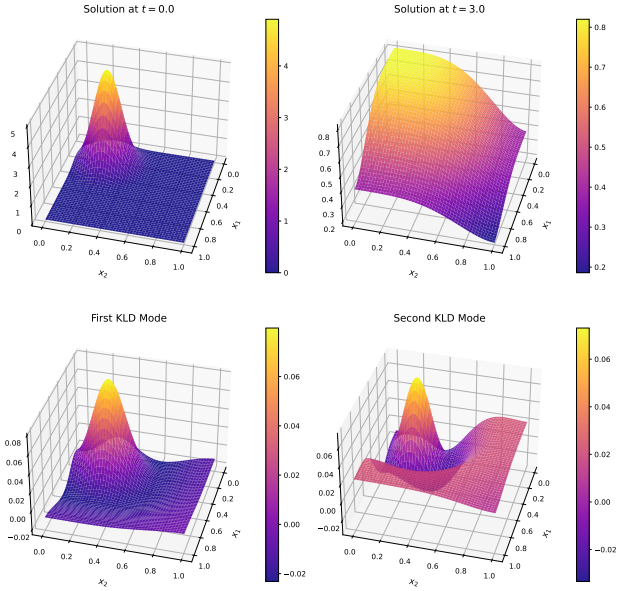

Figure 5. Solution $u(\mathbf{x}, t)$ of the Advection-Diffusion-Reaction system and the first two KLD modes.

and $\mathcal{A}_{\text{old}}$ are 3.31×10^{-6} and 3.81×10^{-5} respectively. The maximum reward obtained using GA was found to be 0.038. While the result from DQN with $\mathcal{A}_{\text{old}}$ are an order of magnitude greater than those under $\mathcal{A}_{\text{new}}$, they are still quite small compared to the maximum result obtained using GA. The training loss decays and stabilizes towards the end. However, the mean episode rewards are unstable across multiple runs and architectures, possibly due to feature-rich KLD modes.

4. Conclusions

The study investigated the application of Reinforcement Learning (RL) techniques, specifically Deep Q-Learning (DQN) and Monte Carlo Tree Search (MCTS), to address the Optimal Experimental Design (OED) problem of sensor placement in spatiotemporal models. The OED problem is computationally challenging due to its combinatorial complexity. The study extended prior work (Wang et al., 2020) that used Temporal Difference (TD) learning for OED by employing DQN with experience replay and fixed Q-targets to enhance scalability and generalization. Additionally, the OED problem was framed as a single-player game and explored using MCTS.

The effectiveness of DQN was examined using two different action space formulations: an "old" action space ($\mathcal{A}_{\text{old}}$) allowing large spatial jumps of sensors and a "new" action space ($\mathcal{A}_{\text{new}}$) restricting sensor movement to adjacent grid cells. These DQN approaches were benchmarked against Genetic Algorithms (GA) on three two-dimensional Dis-

tributed Parameter Systems (DPS): the advection equation, the inviscid Burgers equation, and the advection-diffusion-reaction system.

For the advection equation, DQN with $\mathcal{A}_{\text{old}}$ achieved reward distributions close to the maximum reward obtained by GA, suggesting its effectiveness for this type of system. However, DQN with $\mathcal{A}_{\text{new}}$ exhibited instability in training. MCTS, when applied to the advection system, achieved much higher rewards in some instances compared to both DQN and GA, but its performance was inconsistent, with rewards often concentrated around zero. This inconsistency was potentially attributed to the limited computational budget and/or horizon length used for MCTS, which requires further investigation.

In the case of the inviscid Burgers equation, DQN under $\mathcal{A}_{\text{new}}$ performed marginally better than DQN under $\mathcal{A}_{\text{old}}$, but both RL methods yielded significantly lower rewards compared to GA. The training process for this system appeared to suffer from the agent getting stuck in local minima, possibly due to discontinuities in the solution and sharp gradients in the Karhunen-Loève Decomposition (KLD) modes, making the OED problem more difficult for generalization with DQN.

Similar challenges were observed for the advection-diffusion-reaction system. While DQN with $\mathcal{A}_{\text{old}}$ showed an order of magnitude higher mean reward than DQN with $\mathcal{A}_{\text{new}}$, both were considerably lower than the maximum reward obtained with GA. The instability in mean episode rewards across different runs and architectures was hypothesized to be due to feature-rich KLD modes.

Due to time constraints, MCTS could not be extended to the inviscid Burgers equation and the advection-diffusion-reaction system. Therefore, a comprehensive comparison of MCTS across all three systems was not possible. In conclusion, the study demonstrated the potential of DQN for solving the OED problem, particularly with the old action space for simpler hyperbolic systems like the advection equation. However, the performance of DQN was less satisfactory for more complex systems exhibiting shocks or diffusion, suggesting challenges in generalization. MCTS showed promise for the advection equation but requires further investigation to understand its inconsistent performance and scalability. Future work could focus on refining the RL algorithms and exploring different state and action representations to improve performance and robustness across a wider range of spatiotemporal models. Additionally, examining other reward functions could provide insight into the OED landscape across systems.

References

Gerbrands, J. J. On the relationships between svd, klt and pca. *Pattern Recognition*, 14(1–6):375–381, January 1981.

Krause, A., Singh, A., and Guestrin, C. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(8):235–284, 2008.

Mandli, K. T., Ahmadia, A. J., Berger, M., Calhoun, D., George, D. L., Hadjimichael, Y., Ketcheson, D. I., Lemoine, G. I., and LeVeque, R. J. Clawpack: building an open source ecosystem for solving hyperbolic pdes. *PeerJ Computer Science*, 2:e68, 2016. doi: 10.7717/peerj-cs.68.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. Reinforcement learning for combinatorial optimization: A survey, 2020.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Silver, David, e. a. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, pp. 1140–1144, December 2018.

Singh, A. K. and Hahn, J. Determining optimal sensor locations for state and parameter estimation for stable nonlinear systems. *Industrial amp; Engineering Chemistry Research*, 44(15):5645–5659, June 2005.

Wang, Z., Li, H.-X., and Chen, C. Reinforcement learning-based optimal sensor placement for spatiotemporal modeling. *IEEE Transactions on Cybernetics*, 50 (6):2861–2871, June 2020. ISSN 2168-2275.

Yang, C., Zheng, W., and Zhang, X. Optimal sensor placement for spatial lattice structure based on three-dimensional redundancy elimination model. *Applied Mathematical Modelling*, 66:576–591, February 2019.