

A Algorithms Details

In the context of MFGs, Fictitious Play (FP) operates by iteratively computing the best response of a representative agent against the distribution induced by the average of past best responses of the entire population. The discrete-time FP process involves several key steps at each iteration $k = 0 \dots, K$:

1. **Best Response Computation:** An agent finds its best response policy π_k^{BR} against the approximated average population distribution $\bar{\mu}_{k-1}$.
2. **Average Policy Update:** The average policy $\bar{\pi}_k$ is computed by averaging the current best response policy with all previous policies. In particular we have $\forall t = 0, \dots, N_T$

$$\bar{\pi}_{k-1}(\cdot|x, t) = \frac{\sum_{i=0}^{k-1} \pi_i^{BR}(\cdot|x, t) \mu_t^{\pi_i^{BR}}(x)}{\sum_{i=0}^{k-1} \mu_t^{\pi_i^{BR}}(x)}$$

3. **Average Distribution Update:** The average population distribution $\bar{\mu}_k$ is updated by averaging the current population distribution with past distributions. In particular we have,

$$\bar{\mu}_k = \frac{k-1}{k} \bar{\mu}_{k-1} + \frac{1}{k} \mu^{\pi_k^{BR}}$$

It can be seen that $\bar{\mu}_k = \mu^{\bar{\pi}_k}$

At the end of the algorithm, the pair $(\bar{\pi}_K, \bar{\mu}_K)$ represents the Nash equilibrium of the mean-field game problem. In a model-free framework, however, direct analytical computation is not feasible. For this reason, all three steps must be approximated to fully solve the game. While [Algo. 1](#) and [Algo. 2](#) only address parts of this problem, **DEDA-FP** (described in [Algo. 3](#)) provides the complete solution.

Notation. Here, we provide the notation used in the pseudocodes.

- \mathcal{M} represent the policy buffer use to store all the best responses during FP.
- $\mu_t^{N, \mathcal{M}}$ is the empirical distribution at time t , generated by N agents using a policy assigned uniformly at random from the buffer \mathcal{M} .
- $\mu_t^{N, \pi}$ is the empirical distribution, at time t , generated by N agents using the policy $\bar{\pi}$.
- $J_{\mu_0, \mathcal{M}}^N(\pi)$ is the approximated cost function, computed against $N - 1$ trajectories. These trajectories originate from points sampled from the initial distribution μ_0 and subsequently evolve according to $N - 1$ policies sampled uniformly from the buffer \mathcal{M} .
- $J_{\mu_0, \bar{\pi}}^N(\pi)$ is the approximated cost function, computed against $N - 1$ trajectories. These trajectories originate from points sampled from the initial distribution μ_0 and subsequently all evolve according to $\bar{\pi}$.
- $\mathcal{L}_{\text{NLL}}(\theta) = \mathbb{E}_{(t,s,a) \sim \mathcal{M}_{SL}} [-\log \pi^\theta(a|t, s)] = -\frac{1}{M} \sum_{i=1}^M \log \mathcal{N}(a_i; \mu_\theta(s_i, t_i), \sigma_\theta(s_i, t_i))$, where M is the size of the replay buffer \mathcal{M}_{SL} containing all the triples (t, s, a) sampled from the previous policies and μ_θ and σ_θ are the mean and standard deviation predicted by the policy network for the state s_i at time t_i .

Algo. 1 Simple Approach

- 1: **Input:** Initial distribution μ_0 ; population size N ; Number of iterations K .
- 2: **Initialize:** $\theta_0^*, \mathcal{M} := \{\pi_0^*\}$ where $\pi_0^* := \pi^{\theta_0^*}$.
- 3: **for** iteration $k = 1$ to K **do**
- 4: Using **DRL**, find the best response $\pi_k^* := \pi^{\theta_k^*}$ such that:

$$\pi_k^* = \arg \max_{\pi} J_{\mu_0, \mathcal{M}}^N(\pi)$$

- 5: Add π_k^* in \mathcal{M}
 - 6: **end for**
 - 7: **return** \mathcal{M}
-

Algo. 2 Learning the NE Policy

- 1: **Input:** Initial distribution μ_0 ; N_{sa} : number of state-action pairs to collect at every iteration, N : population size in population simulation; number of iterations K .
- 2: **Initialize:** θ_0^* and set $\bar{\theta}_0 = \theta_0^*$ since $(\bar{\pi}^{\bar{\theta}_0} = \pi^{\theta_0^*})$; sample, according to $\pi_0^* := \pi^{\theta_0^*}$, N_{sa} (time)-state-action triples $(0, s, a)$ and define \mathcal{M}_{SL} to store them.
- 3: **for** iteration $k = 1$ to K **do**
- 4: Using **DRL**, find the best response $\pi_k^* := \pi^{\theta_k^*}$ such that:

$$\pi_k^* = \arg \max_{\pi} J_{\mu_0, \bar{\pi}_{k-1}}^N(\pi)$$

- 5: Collect N_{sa} state-action samples of the form (t, s, a) using π_k^* and store in \mathcal{M}_{SL} .
- 6: Train the **NN policy** $\bar{\pi}_k := \bar{\pi}^{\bar{\theta}_k}$ using supervised learning to minimize the categorical loss:

$$\mathcal{L}_{\text{NLL}}(\bar{\theta}) = \mathbb{E}_{(t,s,a) \sim \mathcal{M}_{SL}} \left[-\log \bar{\pi}^{\bar{\theta}}(a|t, s) \right]$$

This NN aims to mimic the behaviour of the average policy $\frac{1}{k}(\pi_0^* + \dots + \pi_k^*)$.

- 7: **end for**
 - 8: **return** $\bar{\pi}^{\bar{\theta}_K}$
-

B Implementation Details

B.1 Time Conditioned Neural Spline Flow

We employ the Neural Spline Flow (NSF) with autoregressive layers [Durkan et al., 2019] as the flow component in our time conditioned normalizing flow.

Neural Spline Flows The key idea in NSF is to transform a simple distribution (like a standard Gaussian) into a complex one using a series of invertible transformations. To make these transformations very flexible and efficient, NSF uses "rational-quadratic splines."

Rational-Quadratic Splines A spline can be seen as a flexible curve made up of pieces. In our case, each piece is a "rational-quadratic" function, which is a ratio of two quadratic polynomials. These functions are smooth and can be easily inverted, which is important for our model. A rational-quadratic spline is defined by a set of $K+1$ knots $\{(x^{(k)}, y^{(k)})\}_{k=0}^K$. The value of the spline at a given x is determined by which interval $[x^{(k)}, x^{(k+1)}]$ it falls into. Letting $\xi = (x - x^{(k)}) / (x^{(k+1)} - x^{(k)})$ represent the normalized position within that interval, the spline segment is:

$$g(x) = \frac{\alpha^{(k)}(\xi)}{\beta^{(k)}(\xi)}$$

where

$$\alpha^{(k)}(\xi) = s^{(k)}y^{(k+1)}\xi^2 + [y^{(k)}\delta^{(k+1)} + y^{(k+1)}\delta^{(k)}]\xi(1-\xi) + s^{(k)}y^{(k)}(1-\xi)^2$$

$$\beta^{(k)}(\xi) = s^{(k)}\xi^2 + [\delta^{(k+1)} + \delta^{(k)}]\xi(1-\xi) + s^{(k)}(1-\xi)^2$$

and $s^{(k)} = (y^{(k+1)} - y^{(k)}) / (x^{(k+1)} - x^{(k)})$ is the slope of the line connecting the knots at the interval's boundaries. $\delta^{(k)}$ represents the derivative of the spline at knot k .

Autoregressive Neural Spline Flows In our implementation, we use the variant of NSF with autoregressive layers. This means that the parameters of the rational-quadratic spline transformation for each dimension of the data are predicted by an autoregressive neural network. Specifically, for each dimension i of the input x , the spline parameters are computed as a function of the previous dimensions $x_{1:i-1}$:

$$\theta_i = \text{NN}(x_{1:i-1})$$

where NN_{AR} denotes an autoregressive neural network. This autoregressive approach allows the model to capture complex dependencies between the dimensions of the data, as the transformation applied to each dimension is conditioned on the values of the preceding dimensions.

Time Conditioning To handle the non-stationary nature of the mean-field distribution, we explicitly condition the Neural Spline Flow on time t . This means that the entire transformation, and specifically the parameters of the rational-quadratic splines, are made dependent on the current time step. In our autoregressive setup, the neural network that predicts the spline parameters (NN in the equation above) not only takes the previous dimensions $x_{1:i-1}$ as input but also the time t . The time variable t is typically concatenated with the input features or fed into the neural network as an additional input, allowing the network to learn time-dependent transformations. This enables the flow to dynamically adjust its shape and density characteristics as time evolves from $t = 0$ to $t = T$, thereby capturing the non-stationary dynamics of the mean-field.

C Numerical Experiments details

This section provides further experimental results and detailed comparisons between our proposed DEDA-FP approach and the benchmark algorithms considered in the main paper.

C.1 Beach Bar Problem

Further numerical results for the Beach Bar problem are shown in Figures 7 and 8.

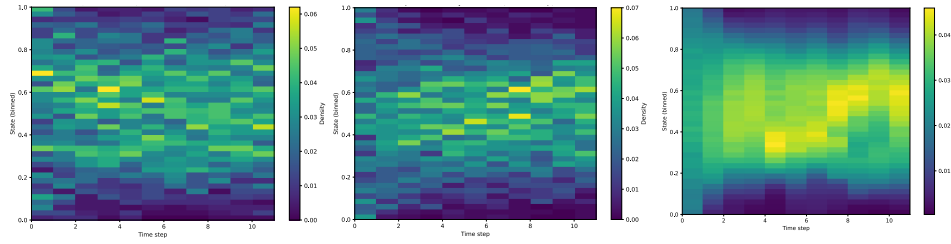


Figure 7: **Comparison of Nash equilibrium distribution heatmaps in the Beach Bar Problem.** From left to right: Algo. 1, Algo. 2, and DEDA-FP (Algo. 3). Thanks to its remarkable speed, DEDA-FP can utilize a high volume of samples (6x times in the displayed figure) for robust distribution approximation, a scale that proves computationally prohibitive for existing benchmarks.

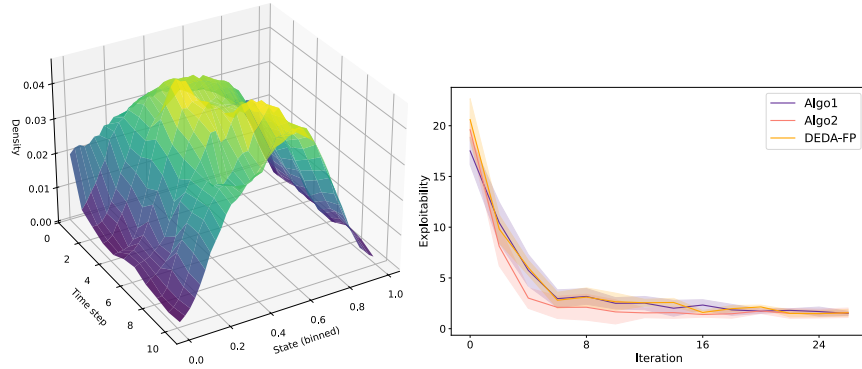


Figure 8: **Beach Bar Problem Results for DEDA-FP.** Left: Nash equilibrium distribution. Right: Exploitability decay comparison across algorithms.

C.2 LQ model

Further numerical results for the 4-rooms exploration problem are shown in Figures 9 and 10.

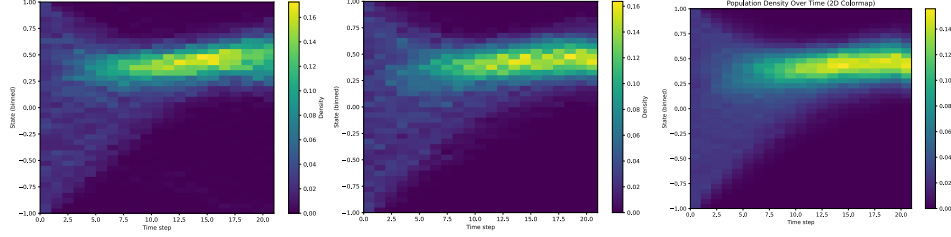


Figure 9: **Comparison of Nash equilibrium distribution heatmaps in the LQ Problem.** From left to right: Algo [Algo. 1](#), Algo [Algo. 2](#), and DEDA-FP ([Algo. 3](#)).

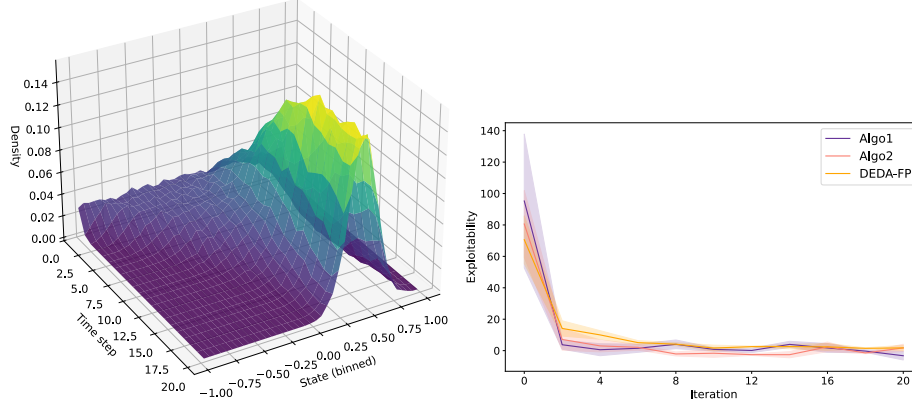


Figure 10: **LQ Problem Results for DEDA-FP.** Left: Nash equilibrium distribution. Right: Exploitability decay comparison across algorithms.

835 C.3 4-rooms exploration

836 Further numerical results for the LQ problem are shown in Figures [15](#) and [16](#).

837 C.4 Market Model

838 Here we present one more example on an environment of a different type, with a financial application.
839 It is a discrete time version of the price impact model in MFG literature, introduced by Carmona and
840 Lacker.¹

841 **Environment:** We consider a market model where $\mathcal{X} = [-5, 5]$ represents the inventory for a stock.
842 The action space $\mathcal{A} = [-1, 1]$ represents the rate of trading for the stock. Each agent controls the
843 inventory for the stock. The dynamics is: $x_{t+1} = x_t + a_t + \epsilon_t$, where $\epsilon_t \sim \mathcal{N}(0, 1)$. The reward is
844 $r(x, a, \bar{a}) = -C_{\mathcal{X}}x^2 - C_{\mathcal{A}}a^2 + hx\bar{a}$, where $C_{\mathcal{X}}$, $C_{\mathcal{A}}$, and h are positive constants, \bar{a} is the mean of
845 the action. At each time t , the representative agent wants to minimize the shares held. In this model,
846 the agent interacts with the distribution of the action instead of the population distribution. The mean
847 field term $hx\bar{a}$ reflects the impact of the action on the price.

848 **Numerical results:** Results are shown in Figures [11](#), [12](#) and [13](#). We observe that traders tend to
849 liquidate their portfolios (given to the x^2 term in the reward function). However, a proportion of
850 agents is incentivized to buy instead of sell due to the interaction term. Moreover, we observe that
851 our model (DEDA-FP) consistently provides a superior representation of the distribution, which is
852 ensured by its efficiency in sampling a large number of agent positions at every time step.

¹René Carmona and Daniel Lacker. A probabilistic weak formulation of mean field games and applications. *Annals of applied probability: an official journal of the Institute of Mathematical Statistics*, 25(3):1189–1231, 2015.

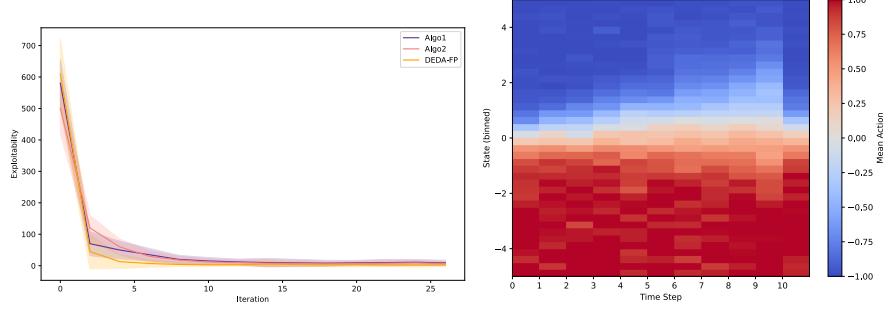


Figure 11: **Market Model Results:** Left: Exploitability decay comparison across algorithms; Right: Nash Equilibrium Policy learned by DEDA-FP

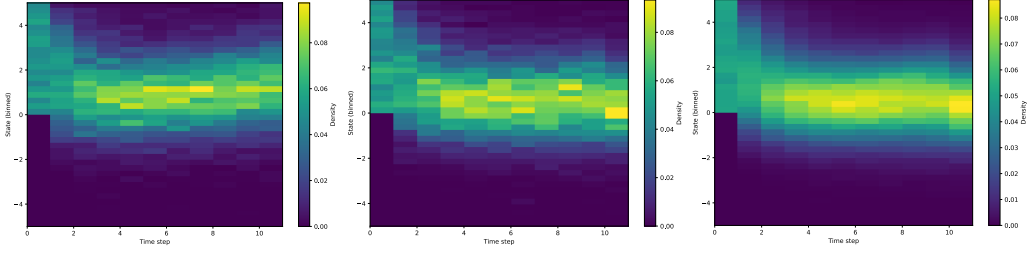


Figure 12: **Comparison of Nash equilibrium distribution heatmaps in the Market Model Problem.** From left to right: Algo [Algo. 1](#), Algo [Algo. 2](#), and DEDA-FP ([Algo. 3](#)).

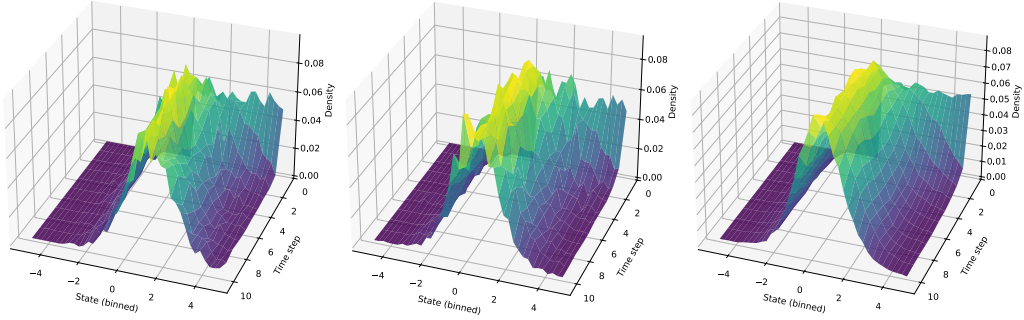


Figure 13: **Comparison of Nash equilibrium distribution in the Market Model Problem ((3D plots)).** From left to right: Algo [Algo. 1](#), Algo [Algo. 2](#), and DEDA-FP ([Algo. 3](#)).

853 D Hyperparameter Sweep

854 We sweep the learning rate over the set $\{3 \times 10^{-2}, 3 \times 10^{-3}, 3 \times 10^{-4}, 3 \times 10^{-5}, 3 \times 10^{-6}\}$ for
 855 Deep RL in [Algo. 1](#) in to the center environment shown in Figure 14. We observe that a learning rate
 856 of 3×10^{-4} yields more stable training and faster convergence. Based on this observation, we adopt
 857 3×10^{-4} for the Deep RL component in [Algo. 2](#) and [Algo. 3](#) as well.

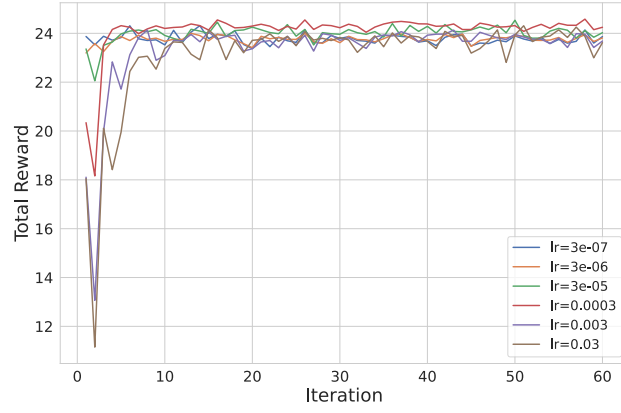


Figure 14: Total reward vs iterations for different learning rate of Deep RL in [Algo. 1](#) in the center environment

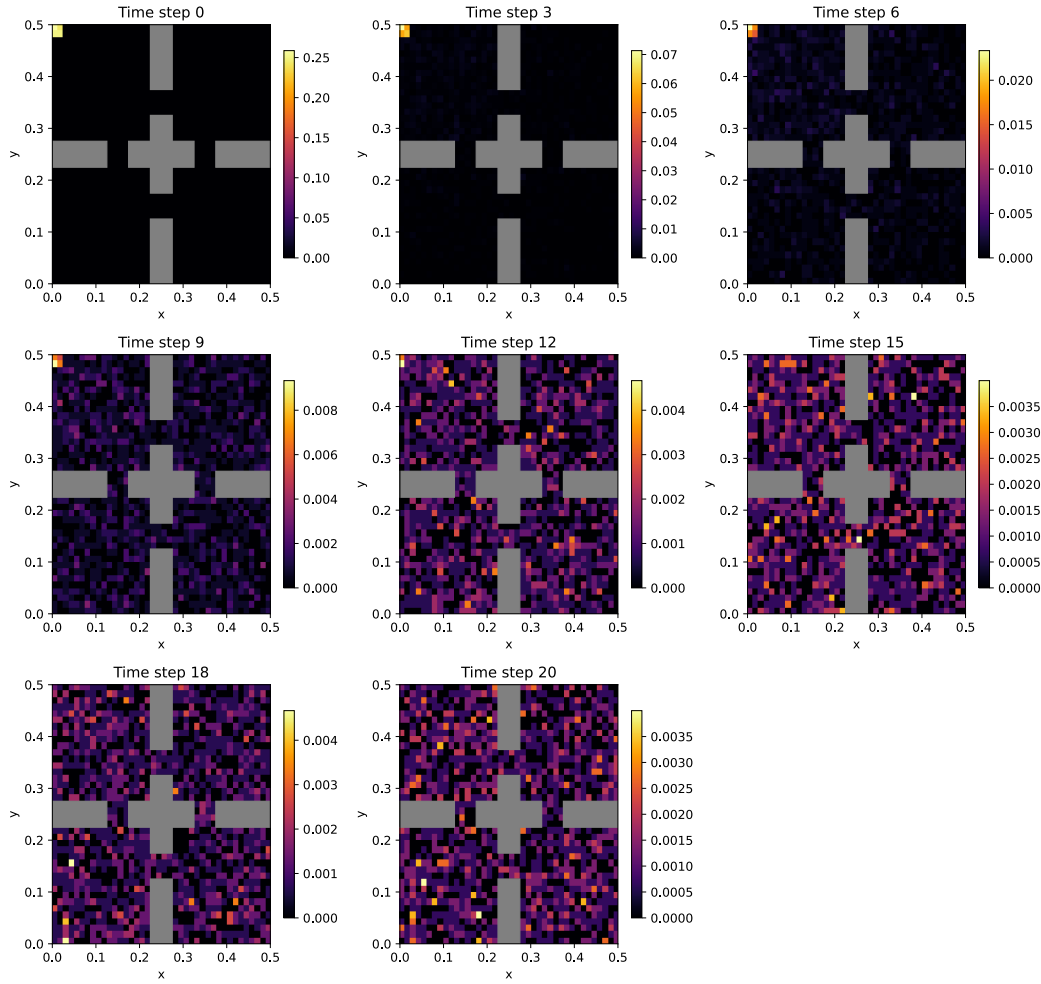


Figure 15: **4 rooms explorations.** Nash Equilibrium mean field flow obtained by [Algo. 2](#) sampling 1500 trajectories

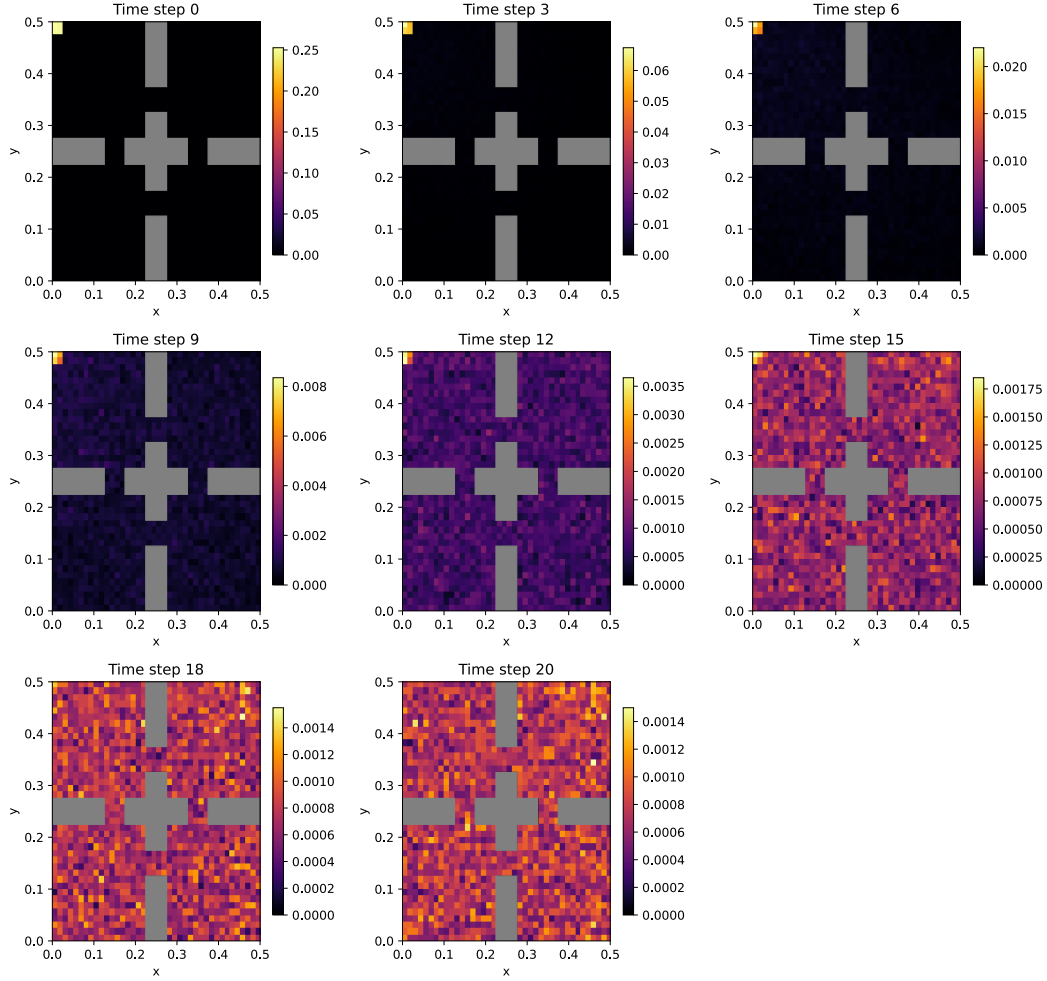


Figure 16: **4 rooms explorations.** Nash Equilibrium mean field flow obtained by [Algo. 3](#) sampling 8000 trajectories $10\times$ faster than [Algo. 2](#) and [Algo. 1](#).