Petroleum Science 20 (2023) 261–276

Contents lists available at ScienceDirect

Petroleum Science

journal homepage: www.keaipublishing.com/en/journals/petroleum-science

Original Paper

Evolutionary-assisted reinforcement learning for reservoir real-time production optimization under uncertainty

Zhong-Zheng Wang [a], Kai Zhang [a, b, *], Guo-Dong Chen [c], Jin-Ding Zhang [a], Wen-Dong Wang [a], Hao-Chen Wang [a], Li-Ming Zhang [a], Xia Yan [a], Jun Yao [a]

[a] School of Petroleum Engineering, China University of Petroleum, Qingdao, 266580, Shandong, China
[b] Qingdao University of Technology, Qingdao, 266520, Shandong, China
[c] The University of Hong Kong, 999077, Hong Kong, China

ARTICLE INFO

Article history:
Received 1 April 2022
Received in revised form
4 August 2022
Accepted 15 August 2022
Available online 23 August 2022

Handling editor: Liang Xue
Edited by Yan-Hua Sun

Keywords:
Production optimization
Deep reinforcement learning
Evolutionary algorithm
Real-time optimization
Optimization under uncertainty

ABSTRACT

Production optimization has gained increasing attention from the smart oilfield community because it can increase economic benefits and oil recovery substantially. While existing methods could produce high-optimality results, they cannot be applied to real-time optimization for large-scale reservoirs due to high computational demands. In addition, most methods generally assume that the reservoir model is deterministic and ignore the uncertainty of the subsurface environment, making the obtained scheme unreliable for practical deployment. In this work, an efficient and robust method, namely evolutionary-assisted reinforcement learning (EARL), is proposed to achieve real-time production optimization under uncertainty. Specifically, the production optimization problem is modeled as a Markov decision process in which a reinforcement learning agent interacts with the reservoir simulator to train a control policy that maximizes the specified goals. To deal with the problems of brittle convergence properties and lack of efficient exploration strategies of reinforcement learning approaches, a population-based evolutionary algorithm is introduced to assist the training of agents, which provides diverse exploration experiences and promotes stability and robustness due to its inherent redundancy. Compared with prior methods that only optimize a solution for a particular scenario, the proposed approach trains a policy that can adapt to uncertain environments and make real-time decisions to cope with unknown changes. The trained policy, represented by a deep convolutional neural network, can adaptively adjust the well controls based on different reservoir states. Simulation results on two reservoir models show that the proposed approach not only outperforms the RL and EA methods in terms of optimization efficiency but also has strong robustness and real-time decision capacity.

1. Introduction

The oil and gas industry has wielded incredible influence in international economics over the past few decades and will remain the backbone of global energy in the coming years. However, considerable fields have entered the maturity phase and the development of new fields is deemed to be an intractable task. Therefore, it is crucial to take efforts to improve the hydrocarbon production efficiency of existing reservoirs, especially in the context of continued volatility in international oil prices.

With the technological developments of digital and smart oilfields, production optimization has attracted increasing attention from the reservoir workers as a systematic workflow to greatly improve hydrocarbon production efficiency (Chen et al., 2010; Chen and Reynolds, 2016; Chang et al., 2020; Xue et al., 2020, 2021). Production optimization aims to obtain the optimal development scheme (e.g., flow rate, location, and pressure) of each active well to maximize economic benefits or accumulative hydrocarbon production. In practice, however, solving such optimization problems is not a trivial task. It is challenged by strong nonlinearity between decision variables and objective function (Yin et al., 2021). In addition, the optimized solution must satisfy the required physical and operational constraints to ensure its feasibility. Furthermore,

* Corresponding author. School of Petroleum Engineering, China University of Petroleum, Qingdao 266580, Shandong, China.
E-mail address: zhangkai@upc.edu.cn (K. Zhang).

https://doi.org/10.1016/j.petsci.2022.08.016

since optimization involves the prediction of future production, numerical simulators are frequently used. Unfortunately, a single simulation run might cost a relatively long time (typically several hours) while a complete optimization generally requires thousands of simulation runs (Zhao et al., 2016). As a result, algorithms with high efficiency are eager to be developed to deal with these challenges.

In recent years, a variety of advanced optimization approaches have been developed. Among them, gradient-based algorithms show extremely high optimization speed since they use gradient information to determine the search direction (Wang et al., 2002; Sarma et al., 2008; Fonseca et al., 2017; Liu and Reynolds, 2020). For production optimization problems, however, they do not have access to the gradient information required from the commercial simulators, making them impractical for real-world applications (Zhang et al., 2021). By contrast, derivative-free algorithms proceed without explicitly calculating the derivatives and thus have better flexibility (Hajizadeh et al., 2010; Ebrahimi and Khamehchi, 2016). Notable examples include the genetic algorithms and particle swarm optimization. Although these methods are rather stable and easy to implement, they suffer from a heavy computation burden since they rely on a large number of simulation runs and therefore do not meet the demand for real-time optimization in the field. To alleviate this issue, major innovations focus on constructing data-driven surrogate models (Zhao et al., 2020; Chen et al., 2021; Kim and Durlofsky, 2021). In these efforts, the surrogate models are trained using simulation input and output and then used as approximation functions to replace the reservoir simulator during optimization. While the computation demand is significantly reduced, it is difficult to guarantee the fidelity of the surrogate models, especially the dimension of the decision variables is large. Although the dimension reduction techniques are available to cope with the so-called 'curse of dimensionality', the modeling errors are significant enough to shadow the benefits of optimization efficiency (Sun, 2020).

The aforementioned production optimization approaches are task-specific. They cannot extract shared patterns between similar tasks and have to solve each task from scratch even though the task changes only slightly. They typically seek to find a deterministic solution and neglect the uncertainty of the subsurface environment, which results in two major limitations. Firstly, the obtained solution is only optimal for the environment under which the optimization is run. Due to the lack of robustness, a performance drop will inevitably occur when deploying the scheme to the real reservoir, known as the 'sim-to-real' gap. Secondly, they are unable to cope with the unknown changes (e.g., one well breaks down at some timesteps) and make adjustments in real-time, leading to the failure of the development scheme. In theory, one approach well suited to tackle these problems is reinforcement learning (RL). Instead of directly optimizing the decision variables, RL features by training a control policy that helps an 'artificial agent' to learn how to act in a dynamic environment (Qiu et al., 2022). As an online learning algorithm, RL can make use of the experience acquired from interacting with the environment, thereby capturing the environmental uncertainty and adapting to various state conditions. Furthermore, the combination of RL with deep neural networks (DNN), known as the deep reinforcement learning (DRL), can directly deal with complex sequential decision problems in an end-to-end fashion (Mnih et al., 2015). Despite these attractive advantages, RL has few applications in the field of production optimization. Prior works either regard it as an alternative to traditional methods or only handle simple decision scenarios (De Paola et al., 2020; Miftakhov et al., 2020; He et al., 2022; Zhang et al., 2022). We contend that two challenges hinder its widespread adoption in this domain. First, most RL approaches lack diverse exploration strategies and converge prematurely to a local optimum in the face of high-dimensional action and state spaces. Such a scenario is likely to be encountered when performing production optimization of actual reservoirs. Secondly, RL algorithms typically suffer from brittle convergence properties while stability and reliability are crucial in practical deployment.

In response to the problems mentioned above, an efficient and robust approach that is capable of reservoir real-time production optimization under uncertainty is proposed in this paper. To be specific, the production optimization problem is carefully formulated as a finite Markov decision process (MDP) in which a high-performance RL algorithm is used to train the optimal control policy that maximizes the long-term net present value (NPV). To address the problems of brittle convergence properties and insufficient exploration in RL, we introduce the population-based evolutionary algorithm (EA), which generates diverse exploration experiences to train the RL agents and promotes stability and robustness due to its inherent redundancy. In turn, the RL agents are periodically injected into the population to participate in the evolutionary process, allowing the evolutionary process to be accelerated as well. Consequently, the proposed approach, which we call evolutionary-assisted reinforcement learning (EARL), has better performance than EA and RL work separately. To further improve the optimization efficiency of the entire process, EARL is designed as a parallel computation architecture. Multiple policies, which are represented by the deep convolution neural networks (CNN), are trained in their respective environments using the data from a shared experience replay buffer. Once trained, the policy learns an explicit mapping relationship between reservoir state and well control and can adapt to new scenarios without additional training. We apply the proposed approach to waterflooding production optimization problems, in which two reservoir models are used to verify its optimization efficiency, real-time decision capacity, and robustness performance.

The rest of this paper is organized as follows. Section 2 reviews the related works. In section 3, we describe the mathematical model of the production optimization problem and reformulate it as an MDP. In section 4, the proposed approach is illustrated in detail. Case studies are discussed in section 5. Section 6 concludes the paper.

2. Related works

Conventional approaches are widely used to solve the production optimization problem, which can be broadly divided into two categories: gradient-based and derivative-free algorithms. For gradient-based algorithms, Wang et al. (2002) used a numerical perturbation method, and Sarma et al. (2008) used the adjoint gradient method to solve production optimization problems. Liu and Reynolds (2020) applied stochastic gradient methods for robust optimization with nonlinear state constraints. Fonseca et al. (2017) proposed an approximate gradient method for production optimization under uncertainty. Although computationally efficient, gradient-based algorithms can be trapped in a local optimum. On the contrary, derivative-free algorithms show strong global optimization ability in the face of high-dimensional variables and multi-modal objective functions. As a type of derivative-free and meta-heuristic optimization method, the evolutionary algorithm (EA) has been widely applied (Hajizadeh et al., 2010; Ebrahimi and Khamehchi, 2016; Wood, 2016). Foroud et al. (2018) evaluated the application of multiple different global optimization algorithms for well control optimization in the Brugge field. Although most of them are rather stable and easy to implement, they bear expensive

computational costs due to a large number of simulation evaluations. This motivates the arrival of the surrogate models. Zhang et al. (2021) used the support vector machine as the surrogate model to replace the reservoir simulator for constrained production optimization. Kim and Durlofsky (2021) proposed a recurrent neural network-based surrogate model to solve the waterflooding optimization problem. Chen et al. (2021) developed a radial basis function surrogate model for high-dimensional expensive optimization problems. Although much progress has been made in improving optimization efficiency, the significant errors introduced by inaccurate surrogate models remain a core challenge for practical applications.

In addition to the studies of optimization efficiency, several research attempts to achieve robust operation by optimizing a solution that maximizes the average NPV over all possible realizations in the uncertainty set, known as robust production optimization. Guo and Reynolds (2018) designed a robust production optimization workflow that estimates the optimal well controls with support vector regression. Zhao et al. (2020) developed a classification-based multi-objective evolutionary algorithm for waterflooding production optimization under geological uncertainty. However, because robust optimization is used to hedge against the worst-case realization of unknown parameters, the resulting solution is frequently too cautious and thus may sacrifice performance on many environmental variants. Compared with these works, our method learns a control policy autonomously by interacting with the environment and therefore naturally captures the environmental uncertainties.

Recent advances in the machine learning community further accelerate the research of production optimization. Among them, RL has been used as a promising solution. De Paola et al. (2020) trained an RL agent that learned the best drilling scheme with the deep Q-network algorithm. Miftakhov et al. (2020) utilized the proximal policy optimization algorithm to optimize the production parameters with the reservoir pressure and water saturation distribution as the observations. He et al. (2022) utilized the proximal policy optimization algorithm to find the optimal drilling scheme for greenfield primary depletion problems. Zhang et al. (2022) developed an effective DRL agent for life-cycle waterflooding production optimization. However, most existing works regard the RL as an alternative to traditional methods and focus on finding the optimal solution for a specific scenario. There is no in-depth research on robustness and real-time decision capacity under uncertainty. Our approach, using RL as the main component, can be seen as an extension of the previous works. By combining the RL with other optimization techniques, we demonstrate that it can be better applied to solving production optimization problems.

Although RL and EA have seen much success in solving challenging production optimization problems independently, synergies between them have not been studied in this domain. Our approach is inspired by research in the field of computer science. Pourchot and Sigaud (2018) proposed CEM-RL, which combined the cross-entropy method and twin delayed deep deterministic policy gradient algorithm, achieving excellent optimization performance on a set of benchmarks. Khadka and Tumer (2018) introduced the evolutionary reinforcement learning. Experimental results demonstrate that the proposed framework significantly outperforms prior RL and EA approaches in solving complex robotic tasks. More famously, Gupta et al. (2021) proposed the deep evolutionary reinforcement learning framework, based on which the embodied intelligence can perform multiple tasks in multiple complex environments. In this paper, we demonstrate that key challenges in production optimization can be well addressed using the advances in machine learning and general-purpose domains.

3. Problem formulation

3.1. Mathematical model of the production optimization

While the proposed approach is general, we apply it to waterflooding production optimization because waterflooding development is the most commonly used secondary oil recovery method and is a hot topic for reservoir engineers. The goal of waterflooding optimization is to maximize economic benefits by adjusting the well controls (e.g., injection/production rates or bottom hole pressures (BHP)) at each timestep. In practical, NPV is generally chosen as the objective function for dynamic optimization, which is mathematically formulated as

$$J = \sum_{n=1}^{N_t}\left\{\left[\sum_{j=1}^{N_P}\left(r_o \cdot q_{o,j}^n - r_w \cdot q_{w,j}^n\right) - \sum_{i=1}^{N_I}\left(r_{w,inj} \cdot q_{winj,i}^n\right)\right]\frac{\Delta t^n}{(1+b)^{t^n/365}}\right\} \tag{1}$$

where N_t represents the number of timesteps in a life-cycle; N_I and N_P denote the number of injection and production wells, respectively; $q_{o,j}^n$ and $q_{w,j}^n$ are the average oil and water production rates of the jth production well over the nth timestep, respectively, STB/d; $q_{winj,i}^n$ is the average water injection rate of the ith injection well over the nth timestep, STB/d; t^n represents the time at the end of the nth timestep; Δt^n is the time interval of the nth timestep, d; r_o, r_w, and $r_{w,inj}$ are the oil revenue, the disposal cost of produced water, and the water injection costs, respectively, USD/STB; b is the annual discount rate.

In this paper, the sequence of well controls at each timestep to be optimized are given as Eq. (2), all of them are continuous variables.

$$\boldsymbol{u} = \left[q_{winj,1}^1, \ldots, q_{winj,1}^{N_t}, q_{winj,2}^1, \ldots, q_{winj,N_I}^{N_t}, \ldots, bhp_{pro,1}^1, \ldots, bhp_{pro,1}^{N_t}, bhp_{pro,2}^1, \ldots, bhp_{pro,N_P}^{N_t}\right] \tag{2}$$

where $q_{winj,i}^t$ denotes the water injection rate of ith injection well at tth timestep; $bhp_{pro,j}^t$ denotes the BHP of jth production well at tth timestep.

Since the water and oil production rates of production wells in the NPV are functions of the system state vector $\boldsymbol{x}$ (includes all the primary variables of the reservoir simulator) and well control vector $\boldsymbol{u}$, we can write the NPV of Eq. (1) as a function of $\boldsymbol{x}$ and $\boldsymbol{u}$. Now, the production optimization problem can be formulated as a general constrained optimization problem

$$\max_{\boldsymbol{u}} J(\boldsymbol{x}, \boldsymbol{u}),\ \boldsymbol{u} \in \mathbb{R}^d \tag{3}$$

subject to

$$g(\boldsymbol{x},\boldsymbol{u})=0,\ \boldsymbol{x}|_{t=0}=\boldsymbol{x}_0 \tag{4}$$

$$\boldsymbol{u}^{\mathrm{lb}}\le\boldsymbol{u}\le\boldsymbol{u}^{\mathrm{ub}} \tag{5}$$

where d represents the number of well controls to be optimized, $d = N_{\mathrm{t}}\times(N_{\mathrm{P}}+N_{\mathrm{I}})$; $g(\boldsymbol{x},\boldsymbol{u})=0$ denotes the reservoir simulation partial differential equations describing the flow of oil, gas, and water; $\boldsymbol{x}_0$ represents the initial state; ub and lb are the specified upper and lower bounds of the corresponding well controls, respectively.

3.2. Production optimization reformulation as Markov decision process

Because waterflooding development strategy consists of a sequence of schemes at several timesteps, the production optimization can be formulated as an MDP, an underlying framework that is widely used to solve sequential decision problems. This MDP is defined by an artificial agent interacting with the environment, including: a state space $\mathscr{S}$, an action space $\mathscr{A}$, a state transition $\mathscr{P}$, and a reward function $\mathscr{R}$. As depicted in Fig. 1, at each timestep t, the agent observes a state $s_t\in\mathscr{S}$ and takes an action $a_t\in\mathscr{A}$ according to its policy $\pi(a_t|s_t)$. Then, the environment evolves to the next state based on state transition $\mathscr{P}$. The agent then receives a scalar reward $r_t\in\mathscr{R}$ and a new state $s_{t+1}\in\mathscr{S}$ for the next timestep $t+1$. This procedure is reiterated for a large number of timesteps. The aim of the agent is to learn an optimal policy π^* that maximizes the expected return, which is defined as

$$\pi^*=\underset{\pi}{\mathrm{argmax}}\,\mathbb{E}\left[\sum_{t=0}^{T}\gamma^t r_t\right] \tag{6}$$

where $\gamma\in[0,1]$ is the discount factor to balance the future reward against the immediate reward; T is the time horizon of the given task.

We now turn to the production optimization problem. The agent can be regarded as the controller which can adjust the production schemes according to the latest reservoir states. The environment is represented by a reservoir simulator that provides the real-time reservoir state and rewards feedback. More specifically, the state, action, and reward function are expressed as follows.

State. The design of the state is supposed to consider both the observability of the system and the diversity of variables. It can be the current situation returned by the environment or any future situation. The selected state variables must provide sufficient information related to the reward function for the agent to understand current situation and take corrective actions. For production optimization problems, the states can be the dynamic information of the wells (e.g., open/close and water cut), the cumulative information of the field (e.g., formation pressure and oil/water production), and temporal volumetric information (e.g., saturation and pressure distribution). In this paper, we choose temporal volumetric information as the state since it provides the richest features. Therefore, the state $\boldsymbol{s}_t$ at the tth timestep is given by

$$\boldsymbol{s}_t=\left[s^t_{\mathrm{o},1},\cdots,s^t_{\mathrm{o},i},\cdots,s^t_{\mathrm{o},n};p^t_1,\cdots,p^t_i,\cdots,p^t_n\right]\in\mathscr{S} \tag{7}$$

where $s^t_{\mathrm{o},i}$ and p^t_i are the oil saturation and pressure of the ith grid at the tth timestep, respectively.

Since the pressure and saturation have different numerical scales, we use Min-Max normalization method $(x-\min(x))/(\max(x)-\min(x))$ to normalize them to improve the training accuracy.

Action. The design of the actions depends on the corresponding environment and specific tasks. For production optimization problems, it is natural to define the well controls as the actions, so the action $\boldsymbol{a}_t$ at the tth timestep is given by

$$\boldsymbol{a}_t=\left[bhp^t_{\mathrm{prd},1},\cdots,bhp^t_{\mathrm{prd},i},\cdots,bhp^t_{\mathrm{prd},N_{\mathrm{P}}};q^t_{\mathrm{winj},1},\cdots,q^t_{\mathrm{winj},i},\cdots,q^t_{\mathrm{winj},N_{\mathrm{I}}}\right]\in\mathscr{A} \tag{8}$$

where $bhp^t_{\mathrm{prd},i}$ and $q^t_{\mathrm{winj},i}$ are the bhp of the ith production well and water injection rate of the ith injection well at the tth timestep, respectively.

Reward function. The design of the reward function is the core of whether an agent can complete the target task. The reward is a

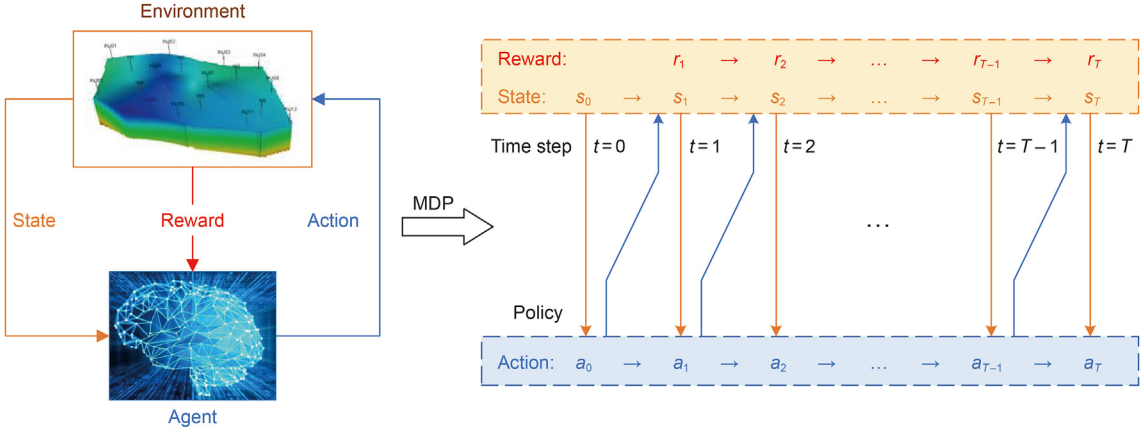

Fig. 1. MDP of waterflooding production optimization problems. The agent and reservoir environment interact at each discrete timestep.

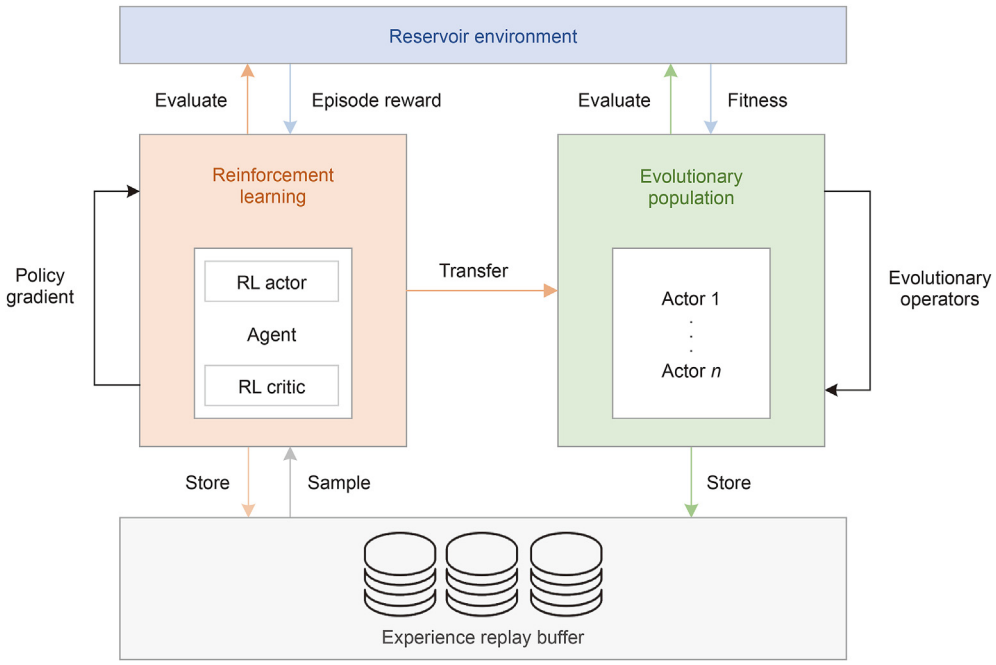

Fig. 2. High-level schematic of the proposed EARL framework. The optimal policy is trained in two interactive processes. Episode reward and fitness denote the evaluation results of the policies provided by RL and EA, respectively. An episode denotes a forward simulation run.

single positive or negative value that assesses whether particular actions are 'good' or 'bad'. The design of the reward should guarantee that the aim of the agent is to maximize Eq. (3). Therefore, the reward r_t at the tth timestep is given by

$$r_t = \eta \cdot \left(\sum_{j=1}^{N_p} \left(r_o \cdot q_{o,j}^t - r_w \cdot q_{w,j}^t \right) - \sum_{i=1}^{N_I} \left(r_{w,inj} \cdot q_{winj,i}^t \right) \right) \cdot \Delta t^n \tag{9}$$

where η is the scaling factor that scales the numerical scale of the reward and facilitates the stability of the training.

4. EARL: evolutionary-assisted reinforcement learning

We propose a hybrid algorithm for real-time production optimization, namely evolutionary-assisted reinforcement learning (EARL). A general framework of EARL is shown in Fig. 2. The optimal policy is obtained in a double-layer training manner: the learning process of RL and the evolutionary process of EA. The key insight is that the synergies between EA and RL enable both the learning process and the evolutionary process can be accelerated. On the one hand, transferring the RL agents into the evolutionary population allows injecting gradient information, which accelerates convergence without losing EA's simplicity. On the other hand, transferring population information into the learning process allows RL agents to leverage diverse exploration experiences and promotes robustness and stability.

4.1. RL-based learning process

Unlike conventional data-driven approaches that rely on large labeled datasets, RL features by an agent that learns an optimal control policy from the experiences obtained by continuous interactions with the environment. Based on the modeled MDP (Section 3.2), we utilize a high-performance DRL algorithm to train the optimal policy for the given real-time production optimization problem.

Soft actor critic (SAC) (Haarnoja et al., 2018) is a state-of-the-art DRL algorithm, which has demonstrated superior performance to solve continuous control tasks. Different from typical RL algorithms that optimize a policy defined in Eq. (6), in SAC, the optimal policy is trained with the goal to maximize the expected return and the policy entropy at the same time, which is given by

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \ \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t \cdot (r_t + \alpha \cdot \mathscr{H}(\pi(\cdot|\boldsymbol{s}_t))) \right] \tag{10}$$

where α is the trade-off coefficient, which balances exploration and exploitation, and

$$\mathscr{H}(p) = \mathbb{E}_{x \sim p}[-\log p(x)] \tag{11}$$

is the entropy term that encourages the agent to explore more action space.

SAC utilizes an actor-critic architecture, which consists of three components: the actor network, the critic network, and the experience replay buffer. The actor network $\pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ parameterized by ϕ represents the policy that maps the state $\boldsymbol{s}_t$ to action $\boldsymbol{a}_t$. The critic network $Q_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)$ parameterized by θ represents the action-value function that maps the state-action pair $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ to Q-value, i.e., expected accumulated reward. The experience replay buffer $\mathscr{D}$ is used to store generated experiences from the interactions with the environment for the training of the actor network and critic network.

More specifically, SAC performs the training process in a policy iteration manner, i.e., iterating multiple times over a policy evaluation step and a policy improvement step. At each iteration, a batch of experiences $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ is sampled randomly from the experience replay buffer $\mathscr{D}$. In the policy evaluation step, the critic network is trained and then used to guide the update of the actor network. Notice that two networks Q_{θ_j}, for $j = 1, 2$ with the same structure are used to alleviate the overestimation of the Q-value. The parameters can be optimized by minimizing the loss function

$$J_Q(\theta_j) = \mathbb{E}_{(\boldsymbol{s}_t,\boldsymbol{a}_t,r_t,\boldsymbol{s}_{t+1})\sim\mathscr{D}}\left[\left(Q_{\theta_j}(\boldsymbol{s}_t,\boldsymbol{a}_t) - y(r_t,\boldsymbol{s}_{t+1})\right)^2\right] \tag{12}$$

with

$$y(r_t,\boldsymbol{s}_{t+1}) = r_t + \gamma\cdot\mathbb{E}_{\boldsymbol{a}_{t+1}\sim\pi_\phi(\cdot|\boldsymbol{s}_{t+1})}\left[\min_{j=1,2}Q_{\theta_{\text{targ},j}}(\boldsymbol{s}_{t+1},\boldsymbol{a}_{t+1}) - \alpha\cdot\log\pi_\phi(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1})\right] \tag{13}$$

where $Q_{\theta_{\text{targ},j}}$ is the target critic network with parameters $\theta_{\text{targ},j}$. The target networks are used to enhance stability during training, and the parameters $\theta_{\text{targ},j}$ are periodically soft updated with

$$\theta_{\text{targ},j} \leftarrow (1-\tau)\theta_{\text{targ},j} + \tau\theta_j,\ \ j = 1,\ \ 2 \tag{14}$$

where $\tau\in(0,1)$ is the smoothing factor.

In the policy improvement step. The actor network is trained by minimizing divergence from the exponential of the soft-Q function. After derivation, the loss function can be written as

$$J_\pi(\phi) = \mathbb{E}_{\boldsymbol{s}_t\sim\mathscr{D},\boldsymbol{a}_t\sim\pi_\phi(\cdot|\boldsymbol{s}_t)}\left[\alpha\cdot\log\pi_\phi(\boldsymbol{a}_t|\boldsymbol{s}_t) - \min_{j=1,2}Q_{\theta_j}(\boldsymbol{s}_t,\boldsymbol{a}_t)\right] \tag{15}$$

The architecture of SAC-based real-time production optimization is shown in Fig. 3. The agent has no prior knowledge of reservoir dynamics. The evolution of the reservoir state and the computation of the reward are provided by the environment. In this process, the policy is represented by the actor network that chooses an action to perform based on the input state. Then, the critic network judges the choices made by the actor network and provides feedback to update its parameters, thus improving the actor network's behavior. By repeating the over and over policy iteration, we eventually save the actor network as the optimal policy.

In practice, the performance RL algorithm is sensitive to the hyperparameter γ. The agent with a small γ cares about short-term returns but may be trapped in a local optimum. The agent with large γ attaches to long-term returns but suffers from an inaccurate prediction. To alleviate this problem, we build an online allocation manager (OAM) to dynamically allocate computational resources for agents with different γ. In particular, the concept of the upper confidence bound (UCB) (Cappé et al., 2013) score U is introduced as the resource allocation criterion, defined as Eq. (16). The agent with the higher U will be assigned more resources in the next iteration.

$$U_i = Q_i + c\cdot\sqrt{\frac{\log\left(\sum_{i=1}^{D}n_i\right)}{n_i}} \tag{16}$$

with

$$Q_i \Leftarrow \omega\cdot score_i + (1-\omega)\cdot Q_i' \tag{17}$$

where Q_i is the estimated value of the ith agent, satisfying the incremental update rule Eq. (17). Q_i' is the past estimated value of the ith agent; $score_i$ is the latest episode reward of the ith agent; ω is the weight coefficient. The square root term is a measure of the uncertainty in the estimate of the ith agent's value. $\sum_{i=1}^{D}n_i$ is the total number of interactions with the environment using all agents. n_i is the number of interactions with the environment using the ith agent. The exploration coefficient c controls the degree of exploration.

During the optimization, each agent is initially assigned the same number of CPU cores to interact with the reservoir environment. The allocation of CPU cores considers not only the current estimated values of agents but the number of times they have been selected. The agents selected less frequently are also given the opportunity to interact with the environment. This operation results in a balance between exploration and exploitation and avoids time-consuming hyperparameter optimization. The pseudocode of the RL-based learning process is provided in Algorithm 1.

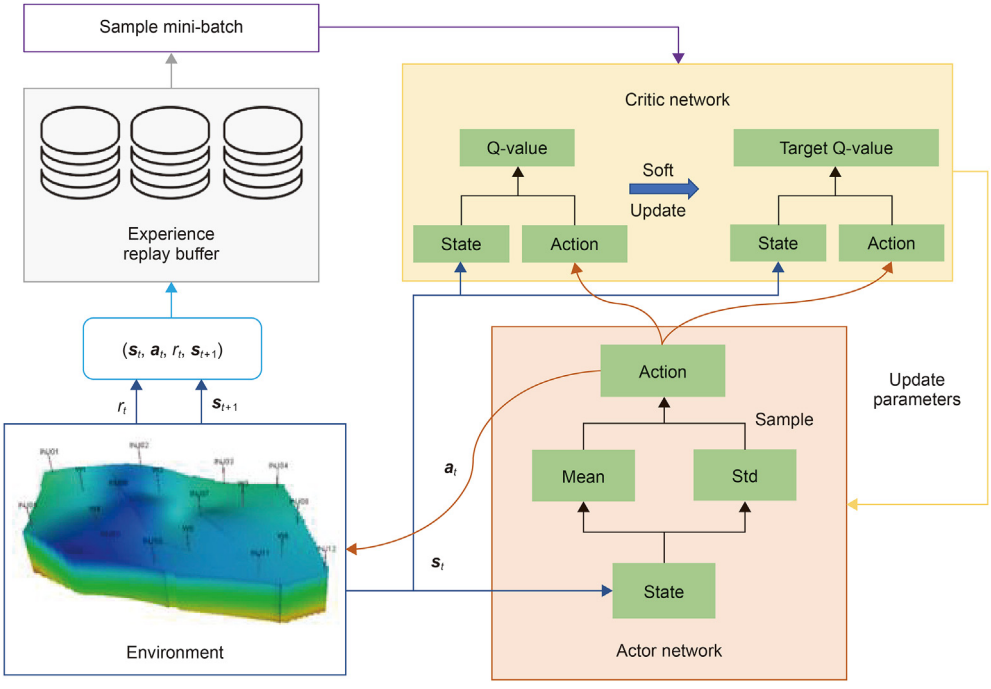

Fig. 3. The SAC-based learning process for real-time production optimization.

Algorithm 1. Pseudocode of RL-based learning process

01: **Initialize:** a group of N_l agents in which each agent has a different γ, an empty allocation set Ω
02: **for** agent i =1 to N_l **do**
03: Update U_i with Eq. (16)
04: Normalize U_i to form a probability distribution P_u
05: Sample from P_u to fill up Ω_i
06: **for** agent i = 1 to N_l **do**
07: **for** index = 1 to Ω_i **do**
08: $score_i \leftarrow$ Policy evaluation (agent) (see Algorithm 2)
09: Update Q_i with Eq. (17)
10: $n_i = n_i + 1$
11: **for** each training step **do**
12: **for** agent i = 1 to N_l **do**
13: Randomly sample a mini-batch of transitions $\left\{\left(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}\right)\right\}$ from $\mathscr{D}$
14: Update the critic network parameters with $\theta_j \leftarrow \theta_j - \lambda_Q \nabla_{\theta_j} J_Q(\theta_j)$, for $j \in \{1, 2\}$
15: Update the actor network parameters with $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi)$
16: Update the target critic network parameters with Eq. (14)
17: **Output:** an updated group of N_l agents

Algorithm 2. Pseudocode of policy evaluation

01: **Input:** policy π
02: **Initialize:** an experience replay buffer $\mathscr{D}$
03: feedback = 0
04: **for** each timestep t **do**
05: Obtain action $\boldsymbol{a}_t = \pi\left(\boldsymbol{s}_t\right)$
06: Execute a_t, receive r_t, and reach a new reservoir state $\boldsymbol{s}_{t+1}$
07: Add the experience $\left(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}\right)$ to $\mathscr{D}$
08: $\boldsymbol{s}_t = \boldsymbol{s}_{t+1}$
09: feedback = feedback + r_t
10: **Output:** feedback

4.2. EA-based evolutionary process

Although RL has successfully improved the performance of real-time production optimization, there are still two shortcomings that hinder its performance. First, RL lacks effective and diverse exploration strategies in the face of large state space and action space. In general, actual reservoir models contain a large-scale number of grids and involve a large number of decision variables, it is difficult to learn a policy that well establishes the mapping relationship between the reservoir states and the well controls. Second, RL suffers from brittle convergence properties, especially when the rewards are uneven during the interaction. The main innovation of the proposed approach is to introduce population-based EA to provide diverse exploration experiences to train the RL agents and promotes stability and robustness.

To achieve the synergies between EA and RL, the individuals in the population must have the same structure as RL agents. Otherwise, the RL agents are unable to utilize the exploration experiences provided by the population and cannot be copied into the population to participate the evolutionary process. To address this problem, a deep Neuroevolution (NE) algorithm (Such et al., 2017) is used as a component of the proposed framework. Unlike methods such as the genetic algorithm and differential evolution that directly optimize a numerical solution, NE parameterizes each individual in the population as a neural network. The selection, crossover, and mutation operators act on the weights of the neural networks to drive the evolutionary process. Thus, NE explores through perturbations in the weight parameters space.

To be specific, the EA-based evolutionary process for real-time production optimization proceeds as follows: a population of actor networks (each actor network represents a policy) is generated with random weights. Note that these actor networks have the same structure as the RL actor networks, but their parameters are updated in different ways. The former uses evolutionary operators, while the latter uses gradient information. For evolutionary actor networks, they are evaluated through a life-cycle of interaction with the reservoir environments. The fitness for each actor network is calculated as the cumulative NPV over the production cycle. A portion of individuals with higher fitness are preserved as elites. The weights of the actor networks are then probabilistically perturbed through the selection, crossover, and mutation operations to create a new generation. Algorithm 3 provides the pseudocode of the EA-based evolutionary process.

Algorithm 3. Pseudocode of EA-based evolutionary process

01: **Initialize:** a population of N_e actor networks (policies)
02: Assess the performance of each policy by computing their episode fitness (see Algorithm 2) and then sort the population by the fitness
03: **for** each evolution iteration **do**
04: Initialize an empty list '*Evolved_policies*' G to fill with offspring for the next generation
05: Add a portion of optimal policies as elites to the new generation instantly
06: **while** the size of G is less than N_e **do**
07: Apply the selection, crossover, mutation operators and keep adding offspring to the G
08: Replace the previous list with the new *Evolved_policies* list, re-evaluate the fitness of each policy and then sort the list by the fitness
09: **Output:** an updated *Evolved_policies* list

4.3. The integrated workflow

The previous two sections described the scenario in which RL and EA independently solve production optimization problems. The core idea of EARL is to integrate them into a single framework to bring two advantages: (1) accelerate the learning process of RL agents and promote stability and robustness of the trained policy using the experiences generated by individuals in the evolutionary population. (2) accelerate the evolutionary process of the population by periodically injecting the RL agents into the population. The detailed implementations are as follows:

EA to RL. Different from conventional EA which only saves the fitness as a feedback signal and does not make use of the information from the interaction process, our approach captures the learning from the intrinsic sequential decision-making process. The experiences of each individual, e.g., which reservoir states the individuals pass through and which well controls they take under these states, are stored in a shared experience replay buffer. These diverse exploration experiences can be randomly sampled by the RL agents to update the parameters of their actor networks and critic networks.

RL to EA. We introduce the concept of sync frequency $\mathscr{M}$, which controls the information flow from the RL agents to the evolutionary population. For example, $\mathscr{M} = 2$ means the actor networks of the RL agents are transferred every two generations. As depicted in Fig. 4, the evolution is run as an outer optimization while the transferred policies are trained with gradient information in an inner loop, which can be considered as a combination of global search and implicit local learning. This mechanism allows the EA's framework to directly use powerful gradient information, accelerating the convergence. The sync results have the following scenarios: if the actor networks transferred are good, they will be selected and extend their influence to the population over the offspring. However, if the actor networks transferred are bad, they will be discarded and gradually eliminated from the population. This result ensures that the effects of sync are positive and will not disrupt the evolutionary process. Algorithm 4 provides the pseudocode of EARL.

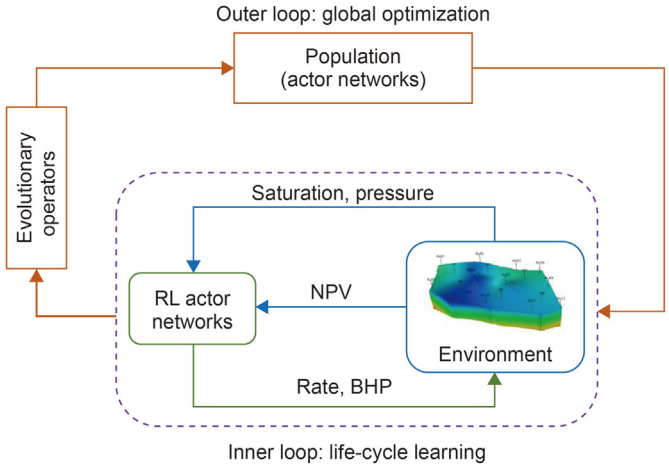

Fig. 4. Policy transfer mechanism between two interactive processes. EA is run as an outer and RL is performed in an inner.

Algorithm 4. Pseudocode of EARL

01: **Initialize:** a population of N_e actor networks, a group of N_l agents, an empty allocation set Ω, an empty experience replay buffer $\mathscr{D}$
02: **Input:** max generations N_{iter}, sync frequency $\mathscr{M}$, save frequency f
03: **for** generation $g = 1$ to N_{iter} **do**
/* Evolutionary process of the population */
04: An updated *Evolved_policies* list $G \leftarrow$ EA-based evolutionary process (see Algorithm 3)
05: **if** $g \bmod f = 0$ **then**
06: Save the optimal policy in the *Evolved_policies* list G
/* Learning process of the RL agents */
07: An updated group of agents $\leftarrow$ RL-based learning process (see Algorithm 1)
/* Periodic transfer */
08: **if** $g \bmod \mathscr{M} = 0$ **then**
09: Copy the actor networks of the RL agents into G to replace the weakest individuals
10: **Output:** optimal policy

4.4. Training details

4.4.1. Neural networks structure

EARL utilizes two types of DNNs to design and optimize the policy: the actor network and the critic network, as shown in Fig. 5. For the actor network, the input is the reservoir state and the output is the well control. In a recent study (He et al., 2022), three-dimensional (3D) reservoir models need to be scaled to a 2D template, then a 2D CNN is used to extract reservoir state features. This operation results in the loss of large raw features because the heterogeneities in the vertical direction are ignored. To address this problem, we use a 3D CNN with a 3D convolution kernel (Ji et al., 2012) to extract information directly from 3D reservoir model. For example, for a 3D model (model size: 116 × 54× 24, the detailed information can be found in the case study section), the input of the actor network is a 5D tensor (*N*,2,24,54,116): 'N' is the batch size, '2' is the number of channels (pressure and oil saturation), '24' is the number of model layers, '54' is the width of the model, and '116' is the length of the model. For the critic network, the input is the combination of the flattened state and action, the output is the Q-value. The critic network is parameterized by fully connected neural networks. The detailed model architecture information about the actor network and critic network can be found in Appendix A.

4.4.2. Training architecture

EARL uses an episodic training fashion in which data are collected by running the simulator with the actor networks. Each episode corresponds to a simulation run that terminates when a fixed timestep has passed. Because of the computational requirements of evolving dynamic reservoir state, the data acquisition rate of the reservoir simulator is significantly slower than that of a typical RL environment. To address this issue, we design a parallel training architecture, as depicted in Fig. 6. At each iteration, the actor networks interact with their respective environments in parallel on allocated CPU cores, and the collected data is stored in a shared experience replay buffer. Then, the critic networks sample the experiences from the buffer to update their parameters and provide gradient information for the update of the actor networks.

5. Case study

In this section, the proposed approach is applied to two production optimization cases of waterflooding reservoir models, including a 2D model and a large-scale 3D model. To verify the effectiveness, EARL is evaluated and compared with the SAC and NE with the same parameters. Meanwhile, to demonstrate the role of the OAM, we use a single agent in the learning process (EARL without OAM), which is a special case of the EARL. Without losing generality, all simulation-based evaluations are performed using the Eclipse^{SM} Reservoir Simulation Software. We implemented all DNN models using the open-source deep-learning library PyTorch (Paszke et al., 2019).

The parameter configurations of EARL are provided as follows: During the evolutionary process, the size of the population N_e is set to 10; the elite rate is set to 0.2. During the learning process, four agents share 10 CPU cores based on the OAM, and their discount factors are set to 0.80, 0.90, 0.95, and 0.99, respectively. The learning rates for the critic networks and the actor networks are set to 5e-4 and 3e-4, respectively. The batch size is set to 128. The smoothing factor τ is set to 0.005. The trade-off coefficient a is set to 0.2. The save frequency f and the sync frequency $\mathscr{M}$ are set to 5. In addition, the weight coefficient ω and the exploration coefficient c in OAM are set to 0.2 and 0.9, respectively.

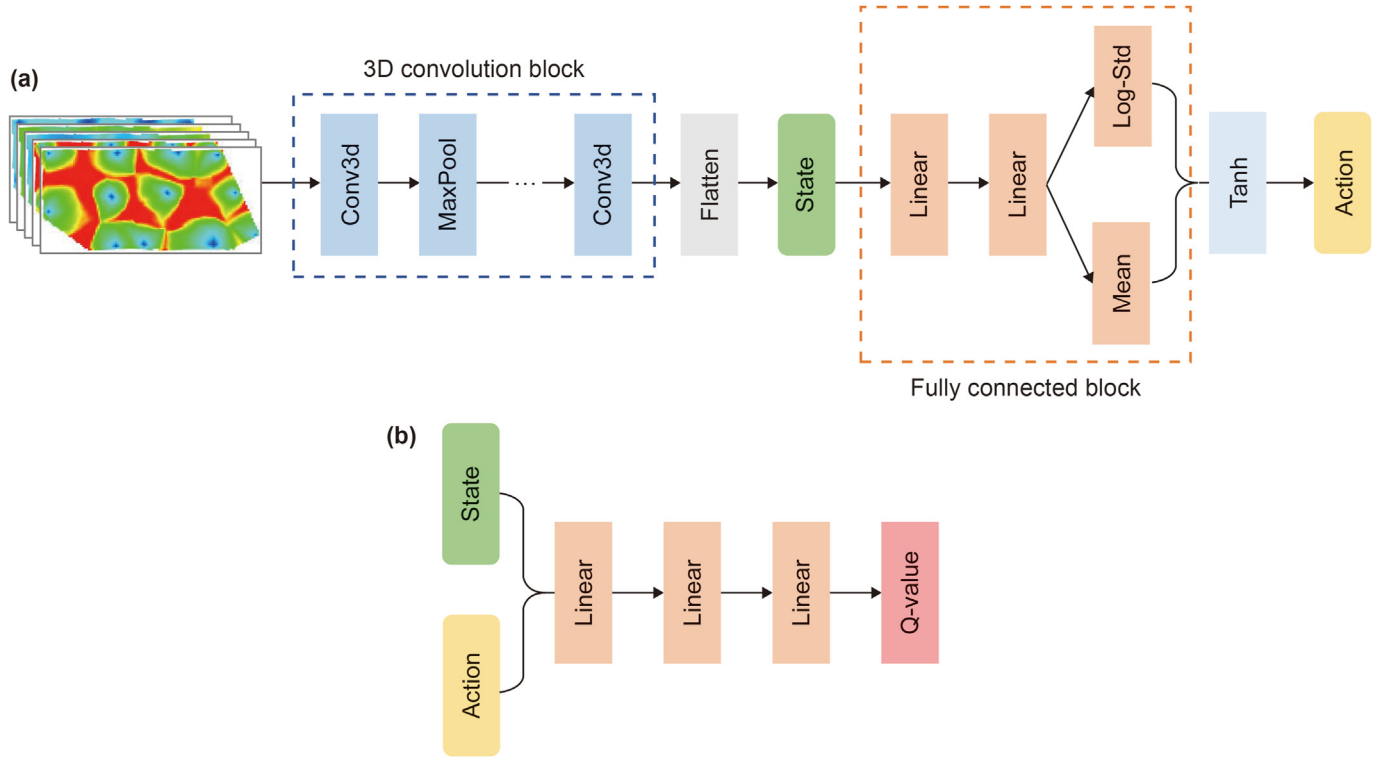

Fig. 5. Network structure. (**a**) Policy network. The input is the reservoir state and the output is the well control. (**b**) Critic network. The input is the combination of flattened state and action, the output is the *Q*-value.

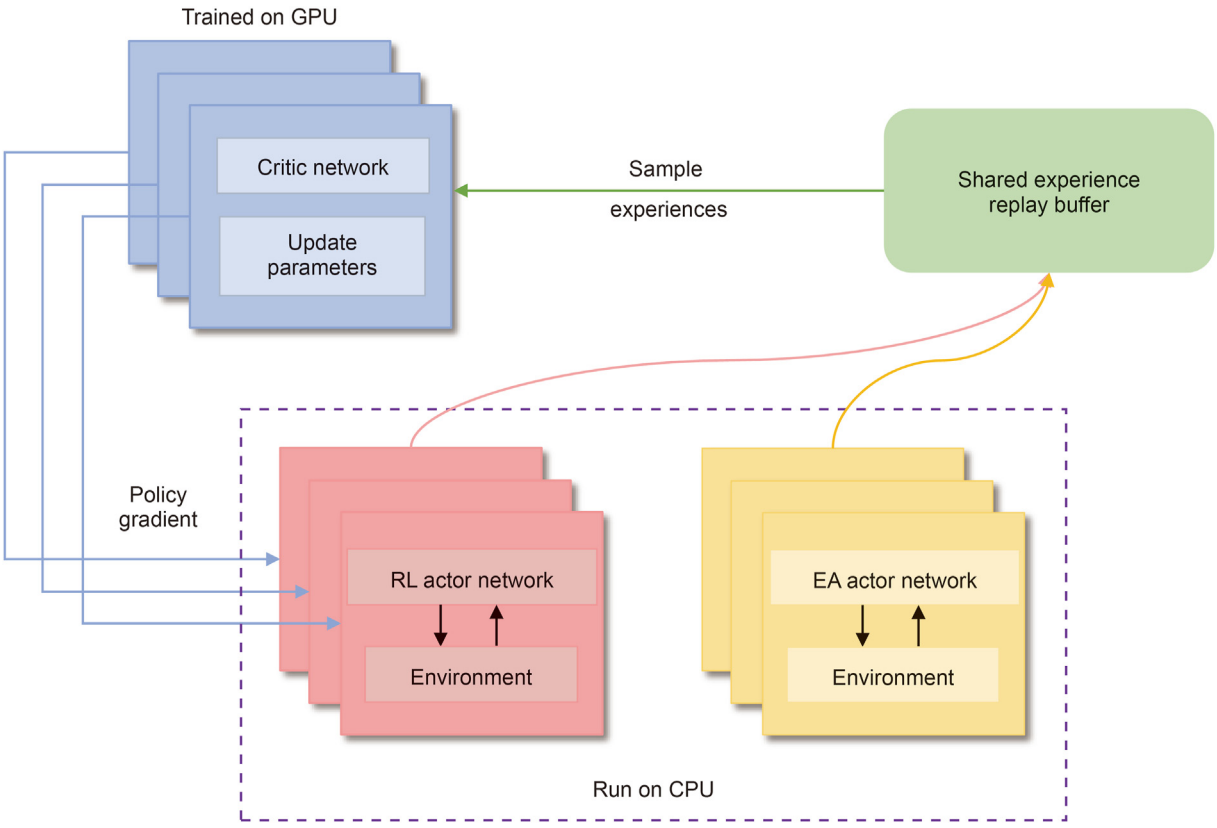

Fig. 6. High-performance and data-efficient architecture for training the optimal policy.

5.1. Case 1: 2D synthetic model

The three channel model is a 2D synthetic reservoir model with three high-permeability channels. This model is developed with four injection wells and nine production wells in a five-spot pattern. The injection wells are under rate control. The allowable upper bound is 1500 STB/d and the lower bound is zero. The production wells are under bhp control. The allowable upper bound is 6000 psi and the lower bound is 3000 psi. The well location and log-permeability distribution of the model are shown in Fig. 7. Typical properties of the model are provided in Table 1. The anticipated production cycle is 1800 days and the interval of each timestep is 180 days, so there are 10 timesteps and the dimension of decision variables is $10 \times (4 + 9) = 130$. The oil revenue is set to 48.0 USD/STB, the water production cost is set to 3.0 USD/STB, and the water injection cost is set to 0.5 USD/STB. Discount rates are not taken into account. The scaling factor is set to 1e-6.

Table 1
Properties of the three channel model.

Properties	Value
Model size	$25 \times 25 \times 1$
Porosity	0.2
Depth, ft	4,800
Viscosity of oil, cP	2.2
Initial pressure, psi	6,000
Compressibility, psi^{-1}	6.9×10^{-5}
Initial water saturation	0.2

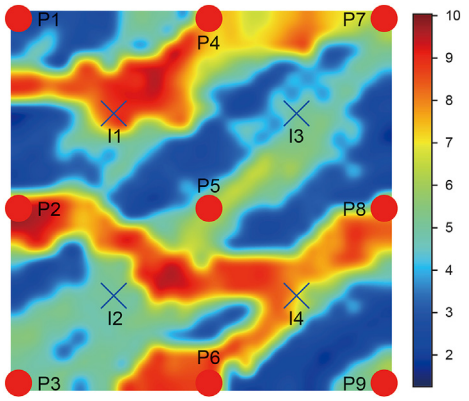

Fig. 7. Well location and log-permeability distribution of the three channel model.

5.1.1. Experimental evaluation on optimization performance

To make fair comparisons, we conducted 10 independent statistical runs with different random seeds and recorded the mean value (solid line) and standard deviation (shadow). The performance of NE, SAC, EARL (without OAM), and EARL are compared based on the same number of simulation runs and their convergence curves are shown in Fig. 8. SAC shows poor optimization performance, which indicates that the agent fails to learn a good policy. This is because the agent trains the policy through generated experience within an episode. But in this case, an episode only has 10 timesteps, the agent is unable to explore the action space sufficiently with a limited number of simulation runs. In contrast, EARL (and EARL without OAM), which combines the global optimization ability of EA, achieves higher NPV and has a smaller variance. Moreover, the addition of OAM brings the synergies between different agents and enables to explore more state and

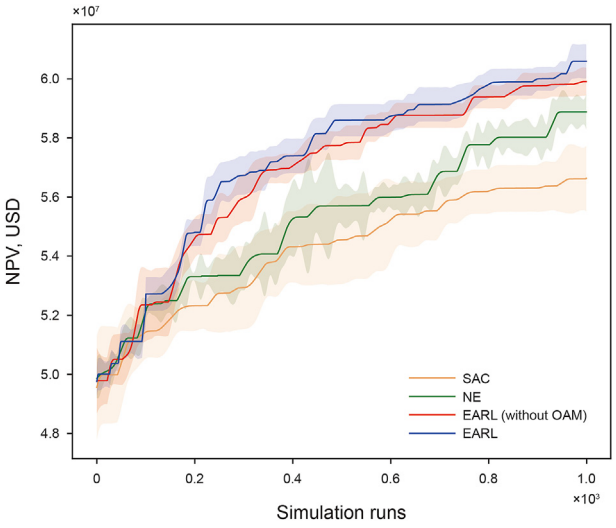

Fig. 8. NPV vs. simulation runs by NE, SAC, EARL (without OAM), and EARL for the three channel model.

action space, further improving the performance of the policy.

From the perspective of oilfield development, achieving an equilibrium displacement of the oil is a key objective. Due to the presence of high permeability channels in the formation, the waterflooding is not uniform. We analyze the quality of the schemes provided by different methods, which is shown in Fig. 9. It is obvious that EARL achieves the best equilibrium displacement of the oil for this plane contradiction problem. The optimal schemes of production wells and injection wells obtained by four different approaches are depicted in Fig. 10 and Fig. 11, respectively.

5.1.2. Experimental evaluation on robustness performance

Robustness remains a major obstacle towards reliable deployment. Traditional production optimization methods are scenario-specific and they fail to generalize to new scenarios even when trained on similar environments with the same optimization objectives. In general, the reservoir models are built by integrating multiple sources of data through history matching. Due to the quality and quantity of the available geophysical data, the model has high uncertainties and thus the discrepancies between the simulation model and the actual reservoir are inevitable. In this section, we evaluate the robustness of the trained policy by

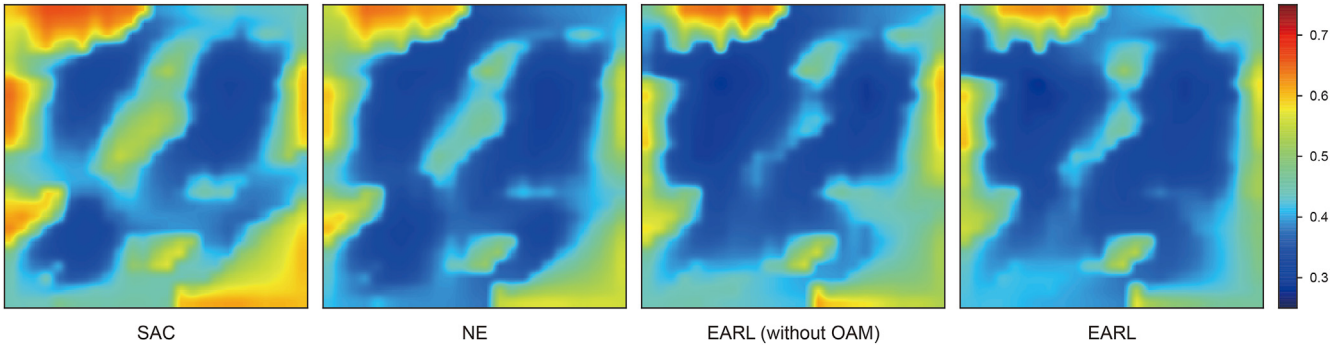

Fig. 9. Comparison of remaining oil saturation distribution after being optimized by different approaches.

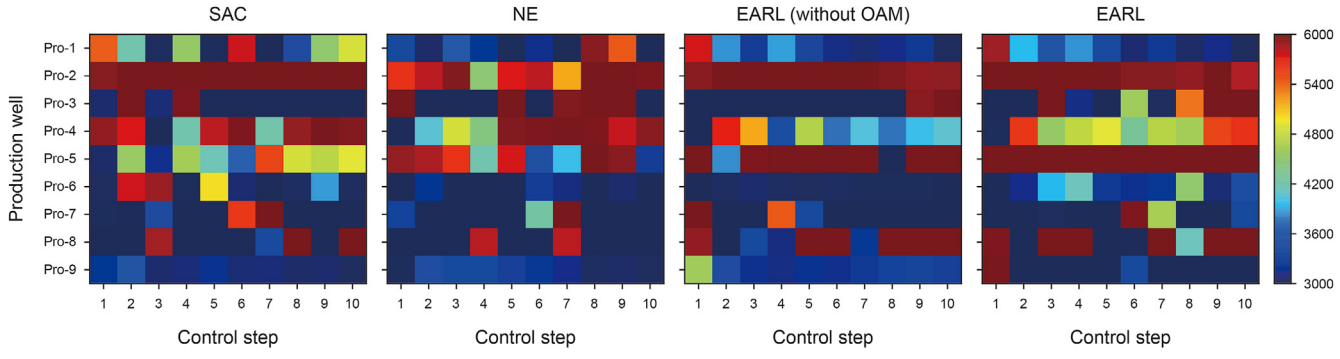

Fig. 10. Optimal BHP controls provided by SAC, NE, EARL (without OAM), and EARL for the three channel model.

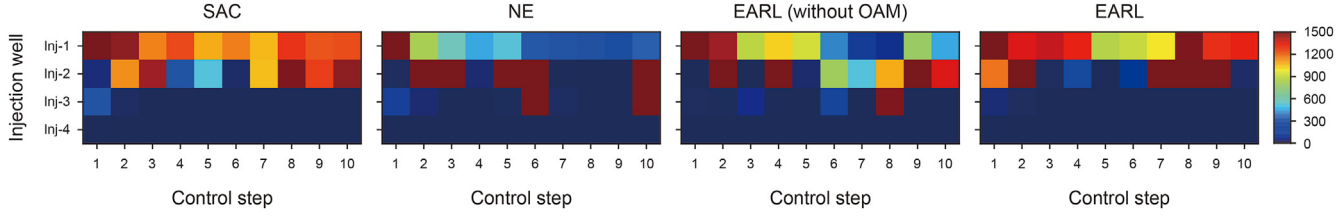

Fig. 11. Optimal rate controls provided by SAC, NE, EARL (without OAM), and EARL for the three channel model.

applying it to three unseen reservoir realizations that have different permeability distributions. Experiments are executed without further tuning of the neural network weights after training. The log-permeability distribution of the three reservoir realizations is shown in Fig. 12.

As a comparison, we retrained a policy from scratch in the current environment. Fig. 13 shows the optimization results of two policies for unseen environments. The experimental results show the trained policy from the original environment can quickly adapt to new environments. Both initial and final performance are significantly improved compared to retraining a policy. This indicates that the trained policy is able to transfer the optimization experiences gained from interacting with the prior environment and use them to efficiently draw inferences and make decisions for similar scenarios.

5.2. Case 2: 3D large-scale model

To further illustrate the effectiveness of the proposed approach, we applied it to a large-scale reservoir model where a certain number of grids are inactive. This model is extracted from a high remaining oil region of an actual reservoir. The block includes six production wells and twelve injection wells. The injection wells are under rate control, the allowable upper bound is 56600 STB/d and the lower bound is zero. The production wells are under bhp control, the allowable upper bound is 1450 psi and the lower bound is 725 psi. Fig. 14 shows the well location and permeability distribution of this model, and the typical properties of the model are shown in Table 2. The anticipated life-cycle is 2400 days and the length of each timestep is 60 days, so there are 40 timesteps in an episode and the dimension of the well controls is $(6 + 12) \times 40 = 720$. High-dimensional decision variables significantly increase the difficulty of optimization. In this case, the oil revenue is set to 80.0 USD/STB, the water production cost is set to 5.0 USD/STB, the water injection cost is set to 0.5 USD/STB, and the annual discount rate is 0%. The scaling factor is set to 1e-5.

5.2.1. Experimental evaluation on optimization performance

We perform 1000 simulation runs to test the performance of four different methods, and the mean and standard deviation values of the NPV derived from 10 independent runs are shown in Fig. 15. We first compare the performance of NE and SAC. In this case, SAC outperforms NE, which is contrary to the result of the

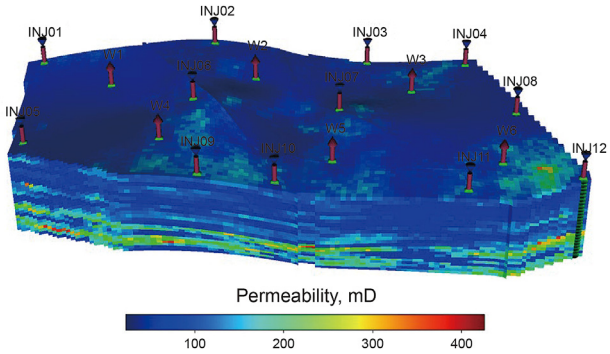

Fig. 14. Well position and permeability distribution of the square model.

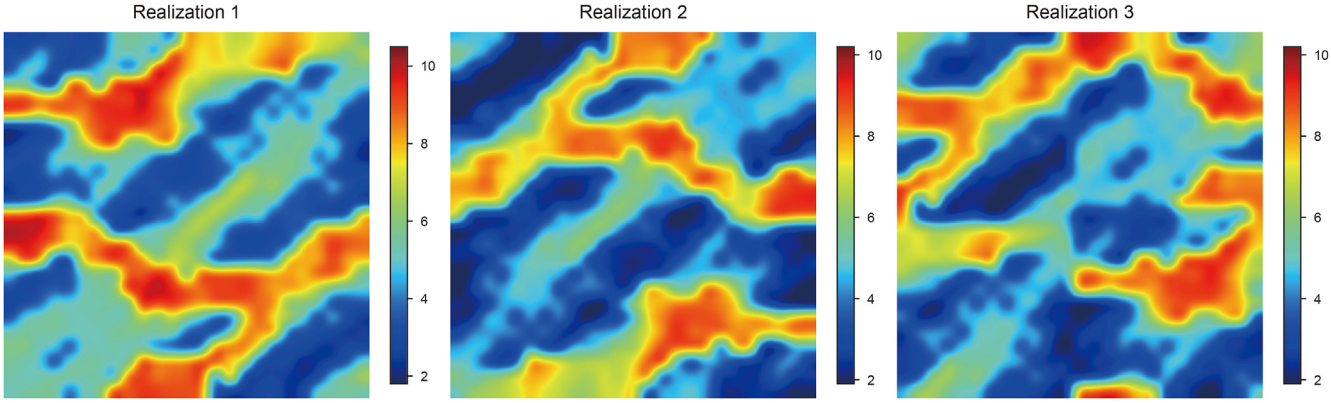

Fig. 12. log-permeability distributions of three reservoir realizations.

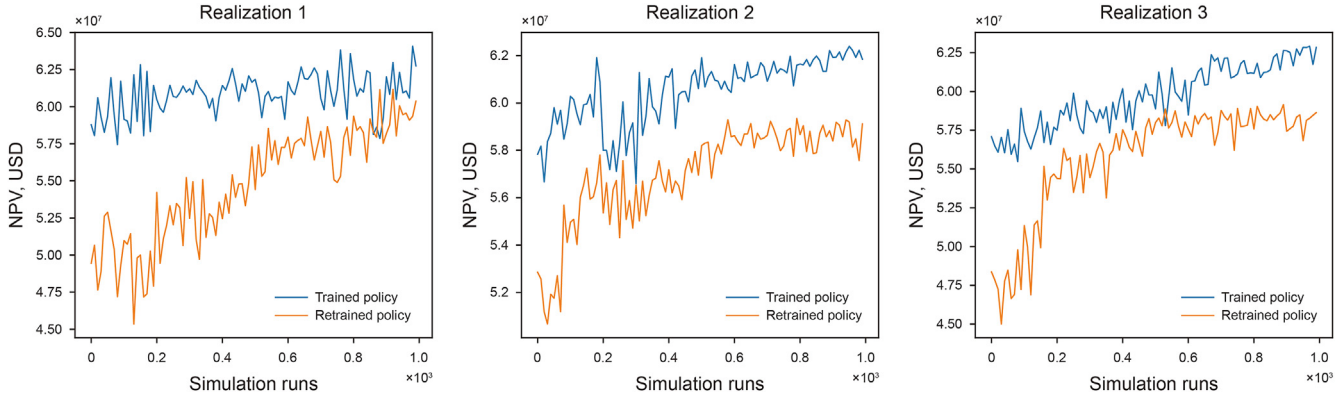

Fig. 13. Comparison of policy's performance under different reservoir realizations.

Table 2
Properties of the square model.

Properties	Value
Model size	116 × 54× 24
Depth, m	900
Viscosity of oil, cP	6.8
Density of oil, kg/m^3	857.8
Compressibility, psi^{-1}	8.2 × 10^{-5}
Initial water saturation	0.32

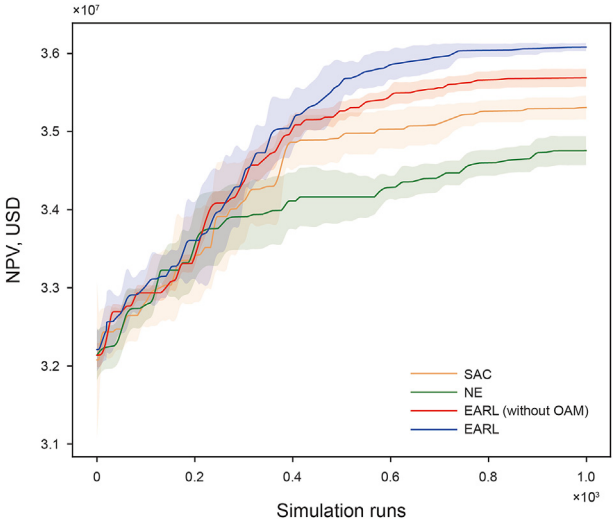

Fig. 15. NPV vs. simulations by NE, SAC, EARL (without OAM), and EARL for the square reservoir model.

three channel model. This is because this model has more wells and timesteps than the three channel model, resulting in a significant increase in the dimensionality of the well-controls. RL methods are more efficient when dealing with multi-step decision tasks since they can make use of the information in the episode. Another conclusion is that EARL has better performance in terms of efficiency and effectiveness compared to NE and SAC. A distinctive feature of waterflooding development is that the NPV is extremely unequal for each timestep. NPV in the late stages of development is significantly smaller than in the early stages. Uneven reward feedbacks make it difficult for the RL agents to learn the optimal policy. In such cases, the global optimization ability of EA plays an important role since it only uses total fitness without leveraging information from intermediate processes. By leveraging the experiences transferred from the population, the RL agents can escape from the local optima. On this basis, the OAM further enhances optimization performance. With different γ, each agent has a different range of exploration. The synergies between different agents lead to a better policy.

Considering the heterogeneity of the reservoir, the differences in water injection displacement can often cause the inter-layer contradiction problem. To address this issue, we employ a CNN with a 3D convolution kernel for the extraction of reservoir state information. Compared with the general feature extraction operation, a 3D convolution kernel can fuse multiple layers of features. Fig. 16 displays the comparison of remaining oil distribution after being optimized by different methods. It can be observed that EARL develops the reservoir more effectively and achieves a higher oil recovery percentage. This is the main reason why EARL attains the highest NPV.

5.2.2. Experimental evaluation on real-time decision capacity

A desirable characteristic of the policy is the ability of functioning in diverse environments, including ones that have never been encountered before. In this section, we investigate whether the trained policy can make real-time adjustments for multiple scenarios, which have similar reservoir environments and the same optimization target (NPV) but differ in the well locations and status. To this end, we evaluate the real-time capacity of the trained policy on two common scenarios.

Scenario 1: an injection well fails to work properly. To be

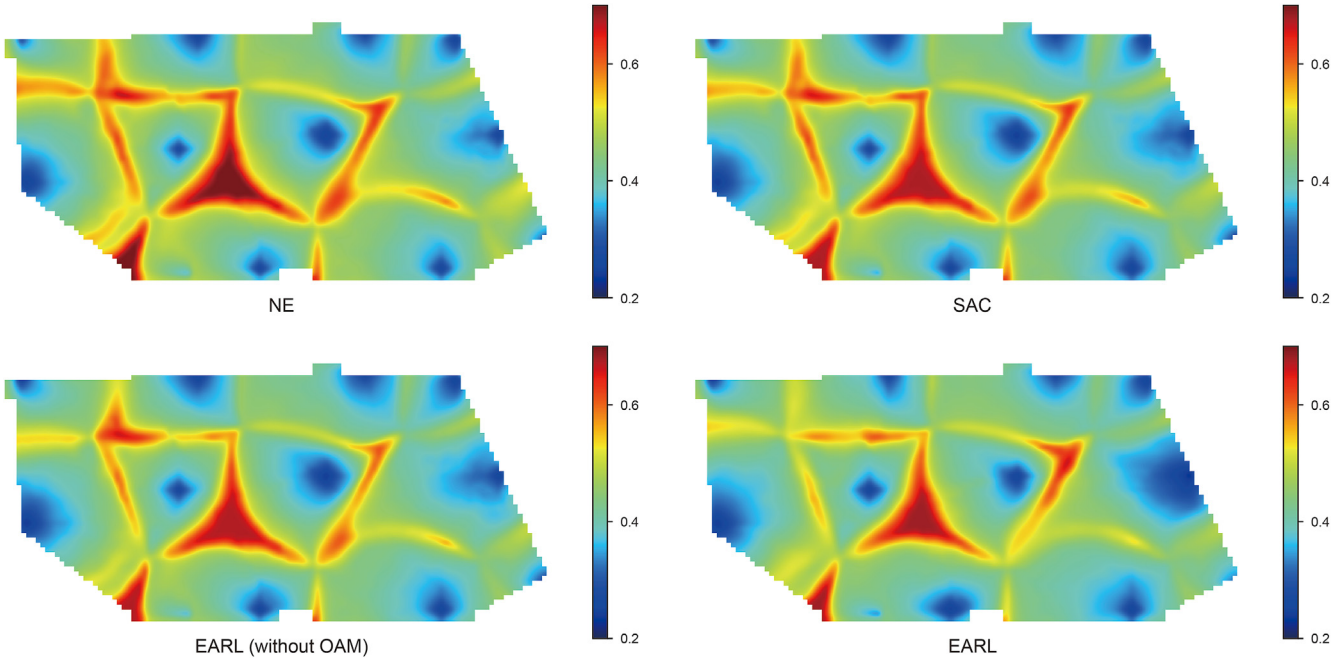

Fig. 16. Comparison of remaining oil distribution after being optimized by different approaches.

specific, the water injection rate of well 7 is set to zero in the first five timesteps of the development. We observe the real-time adjustment capacity of the trained policy. The remaining oil distribution S_o of the development process (a well fails to work during the first five timesteps) is shown in Fig. 17a. As a comparison, the remaining oil distribution $\overline{S_o}$ of the normal development (all the wells work according to the optimal policy) is shown in Figs. 17b and c shows the difference $(S_o - \overline{S_o})$ between these two cases. It can be seen that the trained policy can adaptively make adjustments based on the current reservoir state and eventually achieves a similar oil displacement effect as the optimal development process, despite not being trained in this scenario, further demonstrating the robustness and real-time decision capacity of the trained policy for uncertain environments.

Scenario 2: After developing 10 timesteps, we adjust the well location based on the remaining oil distribution in order to develop the reservoir more effectively. Specifically, we shut in the injection well 7 and well 9, and add two new wells in locations with high remaining oil, which is shown in Fig. 18. Fig. 19a displays the final remaining oil distribution after developing by using the trained policy. For a better comparison, we retrain a policy from scratch by performing a thousand simulation runs, which we consider to be the optimal policy for the current scenario. The remaining oil distribution after developing with this retrained optimal policy is depicted in Fig. 19b. We can see that the trained policy achieves a similar oil displacement effect as the retrained optimal policy.

Fig. 20 shows that the retrained optimal policy does outperform previous trained policy in terms of NPV and cumulative oil production. The reason for this gap is that the previously trained policy encounters new reservoir states that were not experienced during its original training process. However, we emphasize that this gap is within acceptable ranges, especially considering the huge computational costs involved in retraining a policy.

From the above two scenarios, it can be seen that the policy trained with our approach has strong robustness and real-time decision capability. The primary reason behind this is that the information transfer combines the advantages of NE and SAC. SAC provides an explicit structure that maps states to actions, which enables to make full use of powerful gradient information and adjust the well controls based on different reservoir states. However, due to the lack of effective exploration strategies and insufficient exploration of the state space and action space, it is difficult to ensure the performance of the trained policy. The transfer of population information deals with this problem well. During this process, NE provides diverse exploration experiences for the RL agents, which means that two kinds of exploration strategies are utilized in the training process. NE explores through the noise in the parameter space (neural network weights) while SAC explores through the noise in the action space (decision variables). Moreover, it also brings the stability of population-based methods, which can be seen from the variance of multiple independent run results. By continuously interacting with the reservoir environment in both evolutionary and learning ways, the policy generates memorability and adaptively adjusts the well controls by identifying the difference in reservoir states. Therefore, NE and SAC collectively lead to a more effective exploration and improve the performance of the policy.

6. Conclusions

Taking the disadvantages of previous methods applying to production optimization into consideration, this paper proposes a hybrid algorithm, namely EARL, to deal with real-time production optimization under uncertainty. EARL integrates RL and EA into a single framework, achieving better performance than they work separately. This approach enables the training of a robust control policy that can adapt to unseen environments and cope with unknown changes without retraining from scratch, which is fundamentally different from previous methods that can only optimize a solution for a specific scenario. Through the case studies, the proposed approach is proven to have superior optimization efficiency, robustness, and real-time decision capacity.

The main limitation of EARL is that the trained policy only deals with the problems with the same demesion of well controls since the well controls are controlled by the output of the neural network. In addition, the trained policy is only suited to different

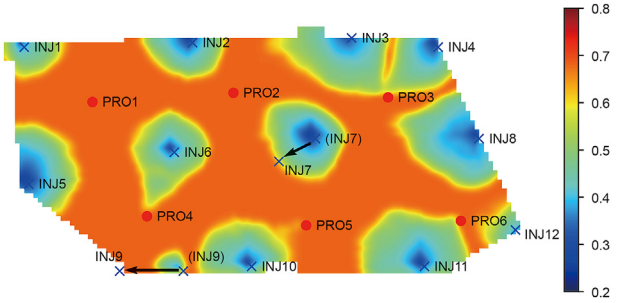

Fig. 18. Position changes of injection well 7 and injection well 9.

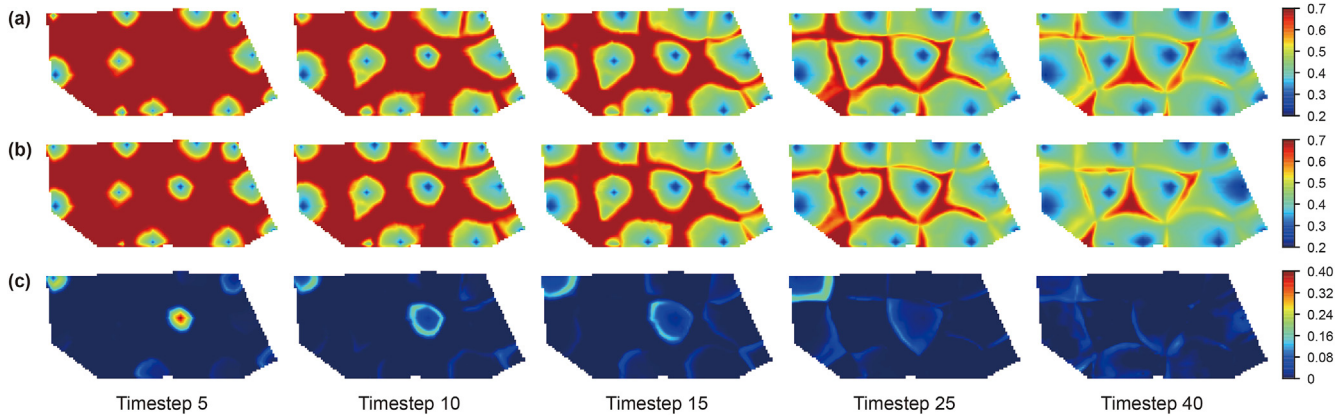

Fig. 17. Comparison of results for scenario 1. (**a**) The evolution of the remaining oil S_o under the scenario where a well fails to work during the first five timesteps. (**b**) The evolution of the remaining oil $\overline{S_o}$ during the normal production process. (**c**) The difference $(S_o - \overline{S_o})$ between the above two cases.

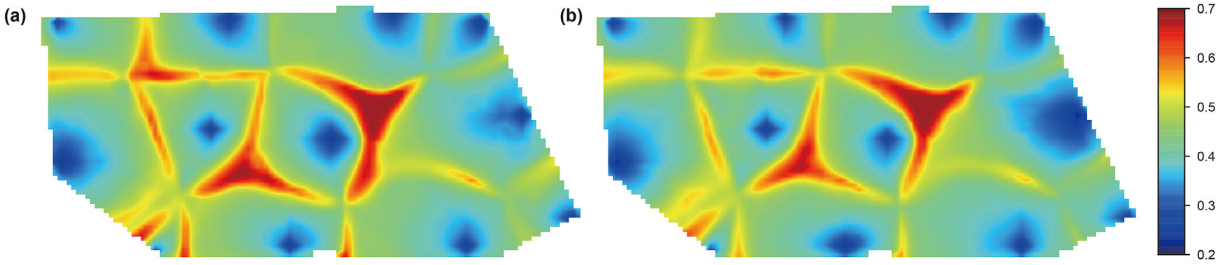

Fig. 19. Comparison of simulation results for scenario 2. Remaining oil distribution after developing with (**a**) the previously trained policy and (**b**) the retrained optimal policy.

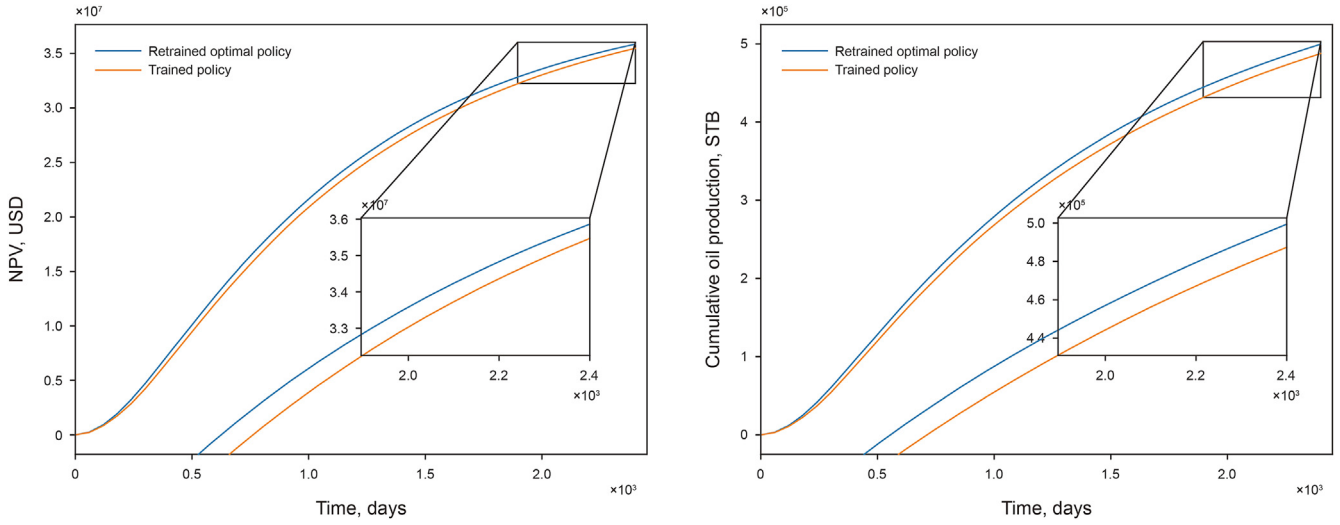

Fig. 20. NPV and cumulative oil production vs. development time for scenario 2.

development scenarios for the same model and cannot be generalized to different reservoir models. From this perspective, this work can be regarded as an initial effort.

Although we study the waterflooding production optimization problems in this paper, EARL is suitable for various multi-step decision tasks involving computationally expensive simulation models in the field of oilfield development, such as well pattern arrangement, directional drilling, and history matching. EARL provides a general framework for further study of these problems. Moreover, EARL utilizes a shared replay buffer that allows the RL agents to use the experiences from EA's population. We wonder whether it is reasonable to directly exploit the actual well control experiences in the field site through such an experience replay buffer when optimizing a practical model. We leave the above issues for future works.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grant 52274057, 52074340 and 51874335, the Major Scientific and Technological Projects of CNPC under Grant ZD2019-183-008, the Science and Technology Support Plan for Youth Innovation of University in Shandong Province under Grant 2019KJH002 , 111 Project under Grant B08028.

Appendix A

Table A1
The model architecture of the actor network. Conv3d, Maxpool3d, and linear are three types of model layers of the PyTorch. ReLU and Tanh are the non-linear activation functions. Action = Normal(μ, σ) represents that the output is sampled from a Normal distribution. N is the batch size of the samples.

Layer	Configuration	Output shape
3D convolution block		
Input	Shape = (N,2,24,54,116)	–
Conv3d	Kernel size=(1,3,3), activation = ReLU	(N,8,24,52,114)
Conv3d	Kernel size=(5,5,5), activation = ReLU	(N,12,20,48,110)
MaxPool3d	Kernel size=(2,2,2), activation = ReLU	(N,12,10,24,55)
Conv3d	Kernel size=(1,5,5), activation = ReLU	(N,16,10,20,51)
Conv3d	Kernel size=(1,5,5), activation = ReLU	(N,24,10,16,47)
MaxPool3d	Kernel size=(2,2,2), activation = ReLU	(N,24,5,8,23)
Conv3d	Kernel size=(1,5,5), activation = ReLU	(N,28,5,4,19)
Conv3d	Kernel size=(1,3,5), activation = ReLU	(N,32,5,2,15)
Flatten		
Flatten	Neurons=(32 × 5× 2× 15), activation = ReLU	(N,4800)
Linear (flattened state)	Neurons = 256, activation = ReLU	(N,256)
Fully connected block		
Linear	Neurons = 256, activation = ReLU	(N,256)
Linear	Neurons = 128, activation = ReLU	(N,128)
Linear (μ)	Neurons = 18, activation = ReLU	(N,18)
Linear ($\ln(\sigma)$)	Neurons = 18, activation = ReLU	(N,18)
Output	Action = Normal(μ, σ), neurons = 18, activation = tanh	(N,18)

Table A2
The model architecture of the critic network. '256 + 18' represents the combination of the flattened state and the action.

Layer	Configuration	Output Shape
Input	Shape = (*N*,256 + 18)	–
Linear	Neurons = 256, activation = ReLU	(*N*,256)
Linear	Neurons = 256, activation = ReLU	(*N*,256)
Linear	Neurons = 128, activation = ReLU	(*N*,128)
output	Neuron = 1	(*N*,1)

References

Cappé, O., Garivier, A., Maillard, O.-A., et al., 2013. Kullback-Leibler upper confidence bounds for optimal sequential allocation. Ann. Stat. 41 (3), 1516–1541. http://www.jstor.org/stable/23566868.

Chang, H., Liu, Y., Lei, Y., et al., 2020. A comprehensive workflow for real time injection-production optimization based on equilibrium displacement. Adv. Geo-Energy. Res. 4 (3), 260–270. https://doi.org/10.46690/ager.2020.03.04.

Chen, B., Reynolds, A.C., 2016. Ensemble-based optimization of the water-alternating-gas-injection process. SPE J. 21 (3), 786–798. https://doi.org/10.2118/173217-PA.

Chen, G., Li, Y., Zhang, K., et al., 2021. Efficient hierarchical surrogate-assisted differential evolution for high-dimensional expensive optimization. Inf. Sci. 542, 228–246. https://doi.org/10.1016/j.ins.2020.06.045.

Chen, S., Li, H., Yang, D., et al., 2010. Optimal parametric design for water-alternating-gas (WAG) process in a CO_2-miscible flooding reservoir. J. Can. Pet. Technol. 49 (10), 75–82. https://doi.org/10.2118/141650-PA.

De Paola, G., Ibanez-Llano, C., Rios, J., et al., 2020. Reinforcement learning for field development policy optimization. SPE Ann. Tech. Conf. Exhib. https://doi.org/10.2118/201254-MS.

Ebrahimi, A., Khamehchi, E., 2016. Sperm whale algorithm: an effective meta-heuristic algorithm for production optimization problems. J. Nat. Gas Sci. Eng. 29, 211–222. https://doi.org/10.1016/j.jngse.2016.01.001.

Fonseca, R.R.M., Chen, B., Jansen, J.D., et al., 2017. A stochastic simplex approximate gradient (StoSAG) for optimization under uncertainty. Int. J. Numer. Methods Eng. 109 (13), 1756–1776. https://doi.org/10.1002/nme.5342.

Foroud, T., Baradaran, A., Seifi, A., 2018. A comparative evaluation of global search algorithms in black box optimization of oil production: a case study on Brugge field. J. Petrol. Sci. Eng. 167, 131–151. https://doi.org/10.1016/j.petrol.2018.03.028.

Guo, Z., Reynolds, A.C., 2018. Robust life-cycle production optimization with a support-vector-regression proxy. SPE J. 23 (6), 2409–2427. https://doi.org/10.2118/191378-PA.

Gupta, A., Savarese, S., Ganguli, S., et al., 2021. Embodied intelligence via learning and evolution. Nat. Commun. 12 (1), 1–12. https://doi.org/10.1038/s41467-021-25874-z.

Haarnoja, T., Zhou, A., Abbeel, P., et al., 2018. Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. Int. conf. mach. learn 1861–1870. https://doi.org/10.48550/arXiv.1801.01290.

Hajizadeh, Y., Christie, M., Demyanov, V., 2010. Comparative study of novel population-based optimization algorithms for history matching and uncertainty quantification: PUNQ-S3 revisited. Abu Dhabi Int. Petrol. Exhib. Conf. https://doi.org/10.2118/136861-MS.

He, J., Tang, M., Hu, C., et al., 2022. Deep reinforcement learning for generalizable field development optimization. SPE J. 27 (1), 226–245. https://doi.org/10.2118/203951-PA.

Ji, S., Xu, W., Yang, M., et al., 2012. 3D convolutional neural networks for human action recognition. IEEE Trans. Pattern Anal. Mach. Intell. 35 (1), 221–231. https://doi.org/10.1109/TPAMI.2012.59.

Khadka, S., Tumer, K., 2018. Evolution-guided policy gradient in reinforcement learning. Proc. Adv. Neural Inf. Process. Syst. 31. https://doi.org/10.48550/arXiv.1805.07917.

Kim, Y.D., Durlofsky, L.J., 2021. A recurrent neural network–based proxy model for well-control optimization with nonlinear output constraints. SPE J. 26 (4), 1837–1857. https://doi.org/10.2118/203980-PA.

Liu, Z., Reynolds, A.C., 2020. A sequential-quadratic-programming-filter algorithm with a modified stochastic gradient for robust life-cycle optimization problems with nonlinear state constraints. SPE J. 25 (4), 1938–1963. https://doi.org/10.2118/193925-PA.

Miftakhov, R., Al-Qasim, A., Efremov, I., 2020. Deep reinforcement learning: reservoir optimization from pixels. Int. Petrol. Technol. Conf. https://doi.org/10.2523/IPTC-20151-MS.

Mnih, V., Kavukcuoglu, K., Silver, D., et al., 2015. Human-level control through deep reinforcement learning. Nature 518 (7540), 529–533. https://doi.org/10.1038/nature14236.

Paszke, A., Gross, S., Massa, F., et al., 2019. Pytorch: an imperative style, high-performance deep learning library. Proc. Adv. Neural Inf. Process. Syst. https://doi.org/10.48550/arXiv.1912.01703.

Pourchot, A., Sigaud, O., 2018. CEM-RL: combining evolutionary and gradient-based methods for policy search. arXiv preprint arXiv:1810.01222. https://doi.org/10.48550/arXiv.1810.01222.

Qiu, D., Dong, Z., Zhang, X., et al., 2022. Safe reinforcement learning for real-time automatic control in a smart energy-hub. Appl. Energy 309, 118403. https://doi.org/10.1016/j.apenergy.2021.118403.

Sarma, P., Chen, W.H., Durlofsky, L.J., et al., 2008. Production optimization with adjoint models under nonlinear control-state path inequality constraints. SPE Reservoir Eval. Eng. 11 (2), 326–339. https://doi.org/10.2118/99959-PA.

Such, F.P., Madhavan, V., Conti, E., et al., 2017. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv preprint arXiv:1712.06567. https://doi.org/10.48550/arXiv.1712.06567.

Sun, A.Y., 2020. Optimal carbon storage reservoir management through deep reinforcement learning. Appl. Energy 278, 115660. https://doi.org/10.1016/j.apenergy.2020.115660.

Wang, P., Litvak, M., Aziz, K., 2002. Optimization of production operations in petroleum fields. SPE Ann. Tech. Conf. Exhib. https://doi.org/10.2118/77658-MS.

Wood, D.A., 2016. Metaheuristic profiling to assess performance of hybrid evolutionary optimization algorithms applied to complex wellbore trajectories. J. Nat. Gas Sci. Eng. 33, 751–768. https://doi.org/10.1016/j.jngse.2016.05.041.

Xue, L., Gu, S.H., Jiang, X.E., et al., 2021. Ensemble-based optimization of hydraulically fractured horizontal well placement in shale gas reservoir through Hough transform parameterization. Petrol. Sci. 18 (3), 839–851. https://doi.org/10.1007/s12182-021-00560-3.

Xue, X., Zhang, K., Tan, K.C., et al., 2020. Affine transformation-enhanced multifactorial optimization for heterogeneous problems. IEEE Trans. Cybern. 52, 6217–6231. https://doi.org/10.1109/TCYB.2020.3036393.

Yin, F., Xue, X., Zhang, C., et al., 2021. Multifidelity genetic transfer: an efficient framework for production optimization. SPE J. 26 (4), 1614–1635. https://doi.org/10.2118/205013-PA.

Zhang, K., Wang, Z., Chen, G., et al., 2022. Training effective deep reinforcement learning agents for real-time life-cycle production optimization. J. Pet. Sci. Eng. 208, 109766. https://doi.org/10.1016/j.petrol.2021.109766.

Zhang, K., Zhao, X., Chen, G., et al., 2021. A double-model differential evolution for constrained waterflooding production optimization. J. Pet. Sci. Eng. 207, 109059. https://doi.org/10.1016/j.petrol.2021.109059.

Zhao, H., Kang, Z., Zhang, X., et al., 2016. A physics-based data-driven numerical model for reservoir history matching and prediction with a field application. SPE J. 21 (6), 2175–2194. https://doi.org/10.2118/173213-PA.

Zhao, M., Zhang, K., Chen, G., et al., 2020. A classification-based surrogate-assisted multiobjective evolutionaryalgorithm for production optimization under geological uncertainty. SPE J. 25 (5), 2450–2469. https://doi.org/10.2118/201229-PA.