Deep Reinforcement Learning Control of a Boiling Water Reactor

Xiangyi Chen and Asok Ray *Fellow, IEEE*

***Abstract*—This paper presents (nonlinear) control system synthesis for a boiling water reactor (BWR) by using artificial intelligence (AI)-based reinforcement learning (RL), where the pertinent algorithm is *Deep Deterministic Policy Gradient* (DDPG). The BWR model, used in this paper, exhibits limit cycling and/or chaotic behavior in different regions of operation. The performance of the RL control system is compared with that of a control system synthesized by the standard $\mathcal{H}_\infty$ theory. The results of comparison show that the RL control system outperforms the $\mathcal{H}_\infty$ control system for disturbance rejection, stability under perturbation, and set-point tracking in a majority of the test cases.**

***Index Terms*—BWR control, Reinforcement learning, Deep Deterministic Policy Gradient**

NOMENCLATURE

a	agent action (scalar-valued in this work)
a_{LB}	lower bound of the action space
a_{UB}	upper bound of the action space
a_i	agent action at step i (*e.g.*, $i = t,\ t+1$)
a'	agent action value generated by a policy depending on the environment state s'
$\tilde{a}_t$	agent action with additive noise at step t
A_t	agent action random variable at step t
$\mathcal{A}$	action space of a MDP
B	minibatch of interactions experiencing
c	excess delayed neutron precursor concentration, normalized to steady-state neutron density
C	neutron precursor concentration
D	coefficient of the Doppler effect of fuel temperature
g_m	gain factor of CRDS servomotor
G_t	return of the rest of an episode after step t
G_m	CRDS servomotor transfer function
G_n	NPMS fission chamber transfer function
k_f	position feedback gain of CRDS
k_n	gain factor of NPMS fission chamber
k_m	reactivity gain factor of CRDS position
K	size of a minibatch of interaction experiences
L	number of tests per policy evaluation
M	total number of episodes used in agent training
n	excess neutron density, normalized to steady-state neutron density
N	neutron density
n^{ref}	set-point of excess neutron density
$\mathcal{P}$	environment dynamics of a MDP
q	value of a state-action pair
$\hat{q}$	approximated value of a state-action pair (s, a)
$\hat{q}'$	approximated value of a state-action pair (s', a')
q_π	value function of a state-action pair under policy π
r	reward value generated by an environment
r_i	reward value generated by an environment at step i, (*e.g.*, $i = t,\ t+1$)
R	replay buffer
R_i	reward random variable at step i (*e.g.*, $i = t,\ t+1$)
R^a_{t+1}	reward random variable component of minimizing control rod movement at step $t+1$
R^c_{t+1}	reward random variable component of exploration restriction at step $t+1$
R^d_{t+1}	reward random variable component of difference between the neutron density and set-point at step $t+1$
$\mathcal{R}$	reward space of a MDP
s	nine-dimensional state vector: $s1, s2, \ldots, s9$ and also (scalar) Laplace transform variable
s_i	environment state value at step i
s'	environment state after one-step state transition
S_t	environment state random variable at step t
S_{t+1}	environment state random variable at step $t+1$
$\mathcal{S}$	state space of a MDP
t	continuous time in Eqs. (1) to (5), (8) to (10); when t is a subscript, it is a discrete-time step
T	total number of interactions per episode
T_e	excess fuel temperature
u	control command
$\boldsymbol{w}$	parameter set for a value neural network
$\boldsymbol{w}^-$	parameter set for a target value neural network
w_a	weight of a reward component for minimizing the control rod movement
w_d	weight of the reward component of a difference between the neutron density and the set-point
W	constraint penalty
x	internal state of NPMS fission chamber
y	readout of NPMS fission chamber
y_i	readout of NPMS fission chamber at step i (*e.g.*, $i = t,\ t+1$)
z	control rod position
$\dot{z}_i$	control rod speed at step i (*e.g.*, $i = t,\ t+1$)
α_1	learning rate for a value neural network

Xiangyi Chen, is with the Department of Nuclear Engineering, Pennsylvania State University, University Park, 16802 PA, USA (e-mail: xxc90@psu.edu).

Asok Ray is with the Departments of Mechanical Engineering, Nuclear Engineering, and Mathematics, Pennsylvania State University, University Park, 16802 PA, USA (e-mail: axr2@psu.edu).

α_2	learning rate for a policy neural network
β	delayed neutron precursors fraction
γ	discount rate
δ	temporal difference error
$\boldsymbol{\theta}$	set of parameters of a policy neural network
$\boldsymbol{\theta}^-$	parameter set of a target policy neural network
κ_i	coefficients in March-Leuba BWR model ($i = 1, 2, 3, 4$)
λ	decay constant for delayed neutron precursors
Λ	neutron generation time
μ	ratio of ξ and ξ_0
ξ	coefficient related to the void reactivity and heat transfer in March-Leuba BWR model
ξ_0	critical value of ξ for BWR instability
π	policy in an agent
π_*	set of optimal policies
ρ	excess overall reactivity
ρ_α	excess void reactivity feedback
ρ_u	excess reactivity inserted by control rod
σ	standard deviation of action Gaussian noise
τ	soft update ratio for target neural networks
τ_m	time constant of CRDS
τ_n	time constant of NPMS fission chamber

ACRONYMS

AI	artificial intelligence
A3C	asynchronous advantage actor critic
BWR	boiling water reactor
CRDS	control rod drive system
DDPG	deep deterministic policy gradient
ILC	iterative learning control
NPP	nuclear power plant
MDP	Markov decision process
MLP	multi-layer perceptron
MPC	model predictive control
NPMS	neutron power measurement system
OC	optimal control
PID	proportional-integral-derivative
POMDP	partially observable Markov decision process
RL	reinforcement learning
RLC	reinforcement learning control
RNN	recurrent neural network
SAC	soft actor critic
TD	temporal difference

I. INTRODUCTION

The boiling water reactor (BWR) nuclear plants are often subjected to large nonlinearities especially when the reactor power is high and the coolant flow rate is low [1]. The plant and its controller together could behave as a non-dissipative system, where large power oscillations may occur. It is possible to encounter instability events occurring in BWR power plants [2], which suggests that the conventional proportional-integral-derivative (PID) controllers are not suitable for BWR control in highly nonlinear regions. To circumvent this instability problem, numerous controller design methods have been investigated; examples are $\mathcal{H}_\infty$ control [3], fuzzy logic control [4], and adaptive predictive control [5], to name a few. All these methods have the potential to be used for synthesis of reactor control systems to accommodate the instability challenge.

In recent years, RL and artificial intelligence (AI) have attracted the attention of both researchers and practitioners for controller design. Investigations in some engineering applications show that a controller synthesized by RL may often exhibit better performance than those synthesized by conventional methods. For example, in smart robotics, RL has shown considerable performance improvement compared to proportional-integral-derivative (PID), model predictive control (MPC), and iterative learning control (ILC) [6].

Research in RL is a vast area in which AI, control theory, and other disciplines have contributed. In early stages, applications of RL in AI and optimal control (OC) have been studied separately where, in the OC area, RL is also commonly called approximate dynamic programming. From the control-theoretic perspectives, research in RL can be classified as *policy iteration*, *value iteration* and *actor-critic* that are usually online and on-policy methods; the value function not only provides the optimal policy but also is a Lyapunov function [7] that establishes the global asymptotic closed-loop stability [8]. While RL using AI is being developed, more innovative methods have used deep neural networks as function approximators; and many algorithms are off-policy methods which use experience replay for sampling efficiency. The problems solved by AI-based RL algorithms are usually discount problems that the control policies are trained to maximize the cumulative discounted return. The benefit of using discounted return is that the algorithm convergence is guaranteed; however, the value function is not a Lyapunov function anymore; therefore, the stability is not guaranteed [9]. For controller design, the closed-loop stability is the foremost consideration. Thus, the closed-loop stability is encoded in the reward function as a performance index, which the AI-based RL control policy is trained to maximize. The two branches, AI and OC, of the RL discipline are also merging, as reported by Bertsekas [10].

In the nuclear engineering community, the theory of RL has been explored for a few applications; apparently, the first application is on autonomous control. The algorithms of deep Q-learning and Asynchronous Advantage Actor Critic (A3C) have been studied to replace the human operators for the tasks of heating-up and power-increase in nuclear power plants [11], [12]; the algorithm of Soft-Actor Critic (SAC) has also been proposed to manage emergency situations in nuclear power plants [13]. The above-mentioned algorithms (*i.e.*, deep Q-learning, A3C and SAC) are based on the AI concepts. The second application by Zhong et al. [14] is on optimal control, which uses the RL concept for controller synthesis, where an iteration-based integral RL algorithm is proposed for reactor power tracking control. In this work, the cost function and the control policy are approximated by high-order polynomials. In another work reported by Dong et al. [15], a multi-layer perception (MLP)-based RL controller is proposed for optimizing the thermal power response of the nuclear steam supply system by generating the optimal

set-points. In this work, an MLP-based state-observer is used for system identification and an approximated optimal control is obtained by solving the algebraic Riccati equation. Both RL algorithms in [14], [15] belong to the class of control-theoretic methods. The third application is on an optimization problem of system design [16] which is not related to plant control.

Apparently the application of RL using the state-of-the-art AI tools has not yet been investigated to synthesize a controller for direct reactivity control. Possible reasons are that the analysis and synthesis of such control systems are much more involved than those of the existing methodologies such as $\mathcal{H}_\infty$ control [3]. In addition, there are many hyperparameters of the AI-based RL, which may need to be tuned, and a systematic way to determine the hyperparameters is yet to be developed. Nevertheless a combination of RL-based control and the state-of-the-art AI has a potential of yielding superior dynamic performance, which will be investigated in this paper, as succinctly discussed below.

In the current paper, the concept of *Deep Deterministic Policy Gradient* (DDPG) [17], [18] is applied to synthesize a BWR control system for reactivity control; the DDPG is an online and off-policy method. The synthesis algorithm is model-free and is capable of accommodating system uncertainties; therefore, the algorithm is suitable for both linear and nonlinear regions of the reactor system. This work serves as an initial research for AI-based RL control of process variables.

The current paper compares the above RL control system with a $\mathcal{H}_\infty$ control system [3]. This comparison is made in standard ways with respect to various system parameters, *e.g.*, by quantifying the performances of: (1) set-point tracking with a step change in the set-point, and (2) disturbance rejection with a step reactivity injection. Beyond that, since the reactor parameters in real-life plants are not static, the transient operations (caused by either set-point step change or set-point ramping) are simulated with system perturbation. These analyses provide an empirical assessment of robustness of the controllers under consideration and test their closed-loop system stability; however, there could be situations, under which the controller may suffer from loss of performance. Nevertheless, these initial assessments suggest a positive potential of RL control using an AI approach, which should draw the attention of the nuclear engineering community for further investigation to have a deeper understanding of both system performance & stability and devise proper methods to enhance the interpretability & predictability of the control system using deep neural network.

This paper is organized in four sections including the present section. Section II presents the BWR model of March-Leuba [19] and its dynamic characteristics as well as the neutronic power measurement and control rod drive systems developed by Suzuki et al. [3]. Section III focuses on the synthesis of RL control and also the performance comparison between the RL control and $\mathcal{H}_\infty$ control. The paper is summarized and concluded in Section IV along with suggested topics of future research.

II. The nonlinear dynamical system of BWR

This section presents a dynamic model of BWR and a model of the neutronic power measurement and control rod-drive systems, on which both RL and $\mathcal{H}_\infty$ control laws have been synthesized.

A. The BWR Nuclear Plant model

The BWR model, developed by March-Leuba [19], [20], has been used by many researchers for instability analysis. The point-kinetics equations, containing the excess neutron density, $n(t)$, normalized to the steady-state neutron density, and the one-group excess delayed neutron precursor concentration, $c(t)$, also normalized to the steady-state neutron density, are

$$\frac{dn(t)}{dt} = \frac{\rho(t)-\beta}{\Lambda} n(t) + \lambda c(t) + \frac{\rho(t)}{\Lambda}, \tag{1}$$

$$\frac{dc(t)}{dt} = \frac{\beta}{\Lambda} n(t) - \lambda c(t), \tag{2}$$

where $n(t)$ and $c(t)$ are defined as: $n(t) \triangleq \frac{N(t)-N|_{ss}}{N|_{ss}}$ and $c(t) \triangleq \frac{C(t)-C|_{ss}}{N|_{ss}}$ respectively, and $|_{ss}$ means at a steady state. The excess fuel temperature, defined as $T_e(t) \triangleq T_e(t) - T_e|_{ss}$, is calculated as

$$\frac{dT_e(t)}{dt} = \kappa_1 n(t) - \kappa_2 T_e(t). \tag{3}$$

The excess void reactivity feedback, defined as $\rho_\alpha(t) \triangleq \rho_\alpha(t) - \rho_\alpha|_{ss}$, is modeled as

$$\frac{d^2\rho_\alpha(t)}{dt^2} + \kappa_3 \frac{d\rho_\alpha(t)}{dt} + \kappa_4 \rho_\alpha(t) = \xi\ T_e(t) \tag{4}$$

The excess overall reactivity feedback is the sum of the feedback of excess void reactivity and fuel temperature,

$$\rho(t) = \rho_\alpha(t) + DT_e(t). \tag{5}$$

where the parameter D represents the Doppler effect of fuel temperature. The pertinent BWR model parameters are listed in Table I, which have been evaluated by functionally fitting the transfer function of the stability test 7N of Vermont Yankee reactor [19].

It is seen from Eq. (1) that the nonlinearity is introduced through the term $\rho(t)n(t)$, where the reactivity $\rho(t)$ depends on $n(t)$. The parameter ξ, which is related to the void reactivity and heat transfer coefficients, determines the reactor stability. At different values of ξ, compared to its critical value $\xi_0 = -3.7 \times 10^{-3}$, the system evolves from a stable fixed point to stable limit cycles, and then to chaos [19]. To see the transitions to various dynamical behaviors, the bifurcation diagram [21], [22] of extrema of $n(t)$ at various values of the parameter $\mu \triangleq \xi/\xi_0$ is plotted in Figure 1. The bifurcation diagram is plotted with 1,400 different values of μ, uniformly distributed from 0.9 to 2.3. The asymptotic behavior of the extrema is simulated for 1,000 s with a step size of 0.001 s for each μ by using the 4th order Runge-Kutta integration method.

TABLE I: BWR model parameters [19]

Parameters	Values	Dimensions
κ_1	25.04	$K \cdot s^{-1}$
κ_2	0.23	s^{-1}
κ_3	2.25	s^{-1}
κ_4	6.82	s^{-2}
ξ_0	-3.70×10^{-3}	$K^{-1} \cdot s^{-2}$
D	-2.52×10^{-5}	K^{-1}
β	5.6×10^{-3}	dimensionless
Λ	4.0×10^{-5}	s^{-1}
λ	0.08	s^{-1}

During the simulation, the initial value of n is set to 0.1 while other state variables are set to 0. The second half (*i.e.*, 500 - 1,000 s) of the trajectory for each μ value is retained, which reflects the asymptotic behavior of the dynamical system; the first half (*i.e.*, 0 - 500 s) of the trajectory for each μ value is discarded because the transients generated during this time interval are due to the (spurious) initial conditions. The points in the trajectory that are greater or smaller than both its last and next time step neighbouring points are recorded as extrema. It is seen from Figure 1 that the system attains a stable fixed point if $\mu \leq 1$. When $1 < \mu \lesssim 1.65$, limit cycles are formed through pitchfork bifurcations. When $\mu \gtrsim 1.65$, (possible) chaotic behavior is observed.

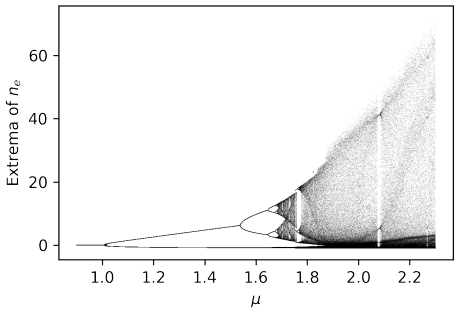


Fig. 1: Bifurcation diagram of extrema of $n(t)$

B. Neutronic power measurement & control rod drive systems

As an extension of the BWR model presented in Subsection II-A, models of the neutronic power measurement system (NPMS) and the control rod drive system (CRDS) developed by Suzuki et al. [3] are used for feedback control. The servomotor of the CRDS is modeled as an integral element:

$$G_m(s) = \frac{g_m}{s}, \tag{6}$$

where the constant g_m is assumed to be 0.4.

The first-order lag model is used to represent the dynamics of the fission chamber and instrumentation electronic circuit:

$$G_n(s) = \frac{k_n}{\tau_n s + 1}, \tag{7}$$

where the gain k_n is assumed to be 1.0, and the time constant τ_n is assumed to be 0.1 s.

The above models of NPMS and CRDS are represented in the state-space setting, where all state variables are scalars, as

$$\frac{dx(t)}{dt} = \frac{n(t) - x(t)}{\tau_n}, \quad y(t) = k_n x(t), \tag{8}$$

$$\frac{dz(t)}{dt} = \frac{u(t) - z(t)}{\tau_m}, \quad \rho_u(t) = k_m z(t), \tag{9}$$

where $x(t)$ represents the internal signal of the fission chamber and instrumentation electronic circuit;
$y(t)$ is the readout value of the fission chamber and instrumentation electronic circuit such that $y(t)$ represents the measured excess neutron density which is derived from the measured neutron flux and its reference steady-state value;
$z(t)$ is the control rod position;
$u(t)$ is the control command (*i.e.*, the desired position); and
$\rho_u(t)$ is the reactivity inserted by the control rod which is proportional to the control rod position.

It is assumed that the position feedback gain, k_f, of the CRDS is 2.5, then the parameters $k_m = \frac{1}{k_f} = 0.4$ and $\tau_m = \frac{1}{g_m k_f} = 1$. A more detailed description of NPMS and CRDS is given by Suzuki et al. [3].

With the control rod reactivity compensation, the total reactivity in Eq. (5) becomes:

$$\rho(t) = \rho_\alpha(t) + DT_e(t) + \rho_u(t). \tag{10}$$

III. Control System Synthesis

This section discusses certain aspects of BWR control system synthesis based on an artificial intelligence (AI)-based reinforcement learning (RL) algorithm, called *deep deterministic policy gradient* (DDPG) [17], as well as the performance comparison between the RL and $\mathcal{H}_\infty$ control systems.

A. Reinforcement learning and optimal control

This subsection presents the basic concepts of the RL and these concepts are used in the construction of the DDPG algorithm; a comprehensive introduction to the RL can be found in Sutton & Barto [23]. In general, the RL scheme is used to select optimal or suboptimal actions (*e.g.*, control commands) via interactions between the agent (*i.e.*, controller) and the environment (*i.e.*, plant) to circumvent the additional task of system identification implicitly, and RL control has the system uncertainty accountable. The framework of the interaction between the agent and the environment is modeled as a *Markov decision process* (MDP) [24] as defined below.

Definition 1 (Markov decision process)**.** *A Markov decision process (MDP) is a quadruple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P})$, where:*

- *$\mathcal{S}$ is a set of states called the state space*
- *$\mathcal{A}$ is a set of actions called the action space*
- *$\mathcal{R}$ is a set of rewards called reward space*
- *$\mathcal{P} : \mathcal{S} \times \mathcal{R} \times \mathcal{S} \times \mathcal{A} \to [0, 1]$ for discrete $\mathcal{S}, \mathcal{A}, \mathcal{R}$, or $\to [0, +\infty)$ for continuous $\mathcal{S}, \mathcal{A}, \mathcal{R}$; and $\mathcal{P}(s', r|s, a)$ is called the environment dynamics, which defines the conditional probability, $\mathbb{P}(S_{t+1} = s', R_{t+1} = r|S_t = s, A_t = a)$, of state and reward values given the immediately preceding state and action*

In Definition 1 as well as in the following contexts, the subscript, t, is time step index instead of time in seconds. Based on Definition 1, the *state-transition probability*, the *reward generating probability* conditioned on a state-action pair and the *expected reward* for a state-action pair can be derived. In the following, these functions are defined for the case of discrete random variables, which can be modified to

the case of continuous random variables. The state-transition probability is defined as

$$p(s'|s,a) \coloneqq \mathbb{P}(S_{t+1}=s'|S_t=s,A_t=a) = \sum_{r\in\mathcal{R}} \mathcal{P}(s',r|s,a). \tag{11}$$

The reward generating probability, given the previous state and action, is defined as

$$p(r|s,a) \coloneqq \mathbb{P}(R_{t+1}=r|S_t=s,A_t=a) = \sum_{s'\in\mathcal{S}} \mathcal{P}(s',r|s,a). \tag{12}$$

The expected reward for a state-action pair is defined as

$$r(s,a) \coloneqq \mathbb{E}[R_{t+1}|S_t=s,A_t=a] = \sum_{r\in\mathcal{R}} rp(r|s,a). \tag{13}$$

The state-transition probability, under the state $S_t = s$ and an action $A_t = a$, determines what is the probability to reach the next state $S_{t+1} = s'$. The expected reward for state-action pair is a function mapping the current state $S_t = s$ and the chosen action $A_t = a$ to the expectation value of the reward to be obtained. Note that the reward is obtained simultaneously as the next state has been generated.

The agent uses the state, S_t, from the environment to determine its action, A_t, via a function called *policy*. The policy maps the state to a deterministic action or a stochastic action drawn from a probability distribution of actions. Once the action, A_t, is applied to the environment, the state, S_t, of the environment transits to a new state S_{t+1} and simultaneously a reward, R_{t+1}, is generated.

The environment dynamics, $\mathcal{P}$, is the rule to generate the new state according to Eq. (11) and the reward according to Eq. (12). The cumulative reward obtained, starting from current step t through an *episode* of the agent and environment interactions, is called *return*, G_t, defined as

$$G_t \coloneqq R_{t+1} + R_{t+2} + \cdots + R_T, \tag{14}$$

where T is total number of interactions per episode that starts at $t = 0$. For AI-based RL algorithms, an augmented return is more commonly used instead of Eq. (14) for both continuing (infinite horizon) tasks and episodic (finite time horizon per episode) tasks:

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{T-t-1} R_T = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k, \tag{15}$$

where T is allowed to be a positive integer or ∞; when T is a positive integer, $\gamma \in [0, 1]$ and when T is ∞, $\gamma \in [0, 1)$; Eq. (15) is called *discounted return*, and γ is called *discount rate*. For continuing tasks, the reward space needs to be bounded for guaranteed boundness of the return.

An *optimal policy* rules the actions that maximize the expected return in each step. The *value function* maps the state-action pair to the expectation value of the return, G_t, averaged over all possible future transitions for the given current state and action. The value function of a state-action pair under a policy, π, is defined as:

$$q_\pi(s,a) \coloneqq \mathbb{E}_\pi[G_t|S_t=s,A_t=a], \tag{16}$$

The set of optimal policies in a policy space is expressed as:

$$\pi_* = \{\pi | q_\pi(s,a) \geq q_{\pi'}(s,a) \text{ for all } \pi' \text{ and } s\in\mathcal{S}, a\in\mathcal{A}\}. \tag{17}$$

The task of a reinforcement learning algorithm is to find one of the optimal (or suboptimal) policies. In the next section, the DDPG algorithm is presented which is used to search the (sub)optimal policy in this paper.

B. Deep deterministic policy gradient algorithm

This subsection presents the principles of the *deep deterministic policy gradient* (DDPG) algorithm [17], [18], and the associated pseudo-code is given in Subsection III-E.

The DDPG algorithm, used in this paper, deals with continuous-action and continuous-space control problems. It is an actor-critic method of reinforcement learning, which consists of a policy function parameterized by a neural network (actor) and a value function parameterized by another neural network (critic). The actor observes the system state and sends the action to the system as a command signal. The critic receives the action generated by the actor and the system state to evaluate the action so as to guide the actor to generate a (possibly) better action with the given state. The actor is denoted by $a = \pi(s;\boldsymbol{\theta})$, where a is the output action, s is the observed state, and $\boldsymbol{\theta}$ is the set of parameters of the neural network. The critic, $q(s,a;\boldsymbol{w})$, with the set of parameters $\boldsymbol{w}$, takes a and s as inputs and predicts the goodness of the action based on the observed state. The goodness is the value defined in Eq. (16), the higher the better. The actor and critic are improved (via updating the parameters $\boldsymbol{\theta}$ and $\boldsymbol{w}$) simultaneously during the interactions with the environment. In general, the actor can generate better actions, and the critic can give more precise evaluation of the actions.

A single interaction with the environment in the MDP setup is a quadruple $(s_t, a_t, r_{t+1}, s_{t+1})$, where s_t and a_t are the observed state and action made at time step t; r_{t+1} and s_{t+1} are the reward and the observed state at the time step $t+1$ after the internal state transition in the environment is completed. After each interaction, two important operations are conducted by the agent as presented in the following two paragraphs.

The first operation is to push the newly generated quadruple to a database called replay buffer. When pushing a quadruple into the replay buffer, the time step index t is disregarded; thus, the instant of a quadruple is denoted as (s, a, r, s').

The second operation is to pull an arbitrarily selected minibatch B of quadruples, $\{(s,a,r,s')_j\}_{j=1}^{K}$, from the replay buffer to train the actor and critic. For each pulled quadruple (s, a, r, s'), the critic predicts the value for (s, a): $\hat{q} = q(s, a; \boldsymbol{w})$, while the actor generates the action for s': $a' = \pi(s';\boldsymbol{\theta})$ and then the critic makes value prediction for (s', a'): $\hat{q}' = q(s', a'; \boldsymbol{w})$. With the results of the above calculations, a temporal difference (TD) error $\delta = \hat{q} - (r + \gamma\hat{q}')$ is obtained which quantifies the prediction accuracy of the

critic. In the TD error, the *TD target* $(r+\gamma\hat{q}')$ is a guess of the true value for (s,a) that is assumed to be more accurate than $\hat{q}$ because it is one step closer to the true value by involving the ground truth of r. So the critic is encouraged to make predictions closer to the TD target to reduce the TD error. This update is achieved by an one-step gradient descent (or any other optimizer) with respect to the mean square TD error as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_1 \nabla_{\boldsymbol{w}} \frac{1}{|B|} \sum_{(s,a,r,s')\in B} \delta^2(s,a,r,s';\boldsymbol{\theta},\boldsymbol{w}), \quad (18)$$

where α_1 is the respective learning rate. The actor should also be updated during the training to generate better actions. Since the objective of the agent is to maximize the expected return which is estimated by the critic, the actor should be updated to generate the actions evaluated by the critic with higher values. Conducting this update means that the actor exploits the evaluation of the critic no matter whether the critic is correct or not. This update is achieved by an one-step gradient ascent (or any other optimizer),

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_2 \nabla_{\boldsymbol{\theta}} \frac{1}{|B|} \sum_{s\in B} q(s,\pi(s;\boldsymbol{\theta});\boldsymbol{w}), \quad (19)$$

where α_2 is the respective learning rate.

It is assumed that the value function can converge during iterations and the converged value function has bounded global maximum; otherwise, the problem cannot be solved by reinforcement learning. The above training process can be summarized as follows:

> Given the current actor, the critic is updated to predict more accurate values through learning from TD errors, while the actor is updated so that the current critic can provide a higher evaluation of the actor. While the update of the actor may alter the TD error because a' is updated along with the actor update even though s could be the same; in this case, the critic needs to learn the new accurate prediction. The driving force behind the above iterations to train an optimal policy is the ground truth r, distributed in the state-action space.

One assumption made above in the TD error is that the TD target $(r+\gamma\hat{q}')$ is a better estimate than $\hat{q}$. This manner of learning a guess from a guess is called bootstrapping [23], which may suffer from bias and instability. The problem can be partially resolved by using a different neural network to evaluate q' so that the target values are constrained to change slowly during iteration [17]. These neural networks are called *target actor* and *target critic* because they are designed to generate the TD target $(r+\gamma\hat{q}')$ by evaluating q'. The target actor and target critic, denoted by $\pi(s;\boldsymbol{\theta}^-)$ and $q(s,a;\boldsymbol{w}^-)$, have the same structure as the actor and critic but have different values of the parameters. In each training step, the parameters of the target actor and target critic are soft-copied from the actor and critic as

$$\boldsymbol{w}^- \leftarrow \tau\boldsymbol{w} + (1-\tau)\boldsymbol{w}^-, \quad (20)$$

$$\boldsymbol{\theta}^- \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta}^-, \quad (21)$$

where the $\tau \in (0,1]$ is a weight that is assigned with a small value to have the target values change slowly during iterations.

It is seen that the target actor and target critic depend on the actor and critic. Therefore, this method cannot resolve the bias problem completely. In practice, the target neural networks are widely used, and show good improvement for the training process. With the target neural networks, the following TD error is used:

$$\delta = q(s,a;\boldsymbol{w}) - (r + \gamma q(s',\pi(s';\boldsymbol{\theta}^-);\boldsymbol{w}^-)). \quad (22)$$

C. Exploration and exploitation

To train the neural networks with better generalization, the state-action space needs to be explored. In other words, the replay buffer should contain a variety of experiences evenly distributed in the state-action space so that a (possible) overfitting can be avoided. However, exploitation can make the learning process faster, where the rich experience around (sub)optimal trajectories often helps the agent to produce better actions when it is on a correct track. To this end, during training, Gaussian noise is added to the actions while the agent interacts with the environment for a trade-off between exploration and exploitation:

$$\tilde{a}_t = \pi(s_t) + noise \quad (23)$$

where $noise \sim \mathcal{N}(0,\sigma^2)$. Since the actuator has operation limitation, the actual action should be limited. Thus the action is clipped by its higher and lower values as:

$$\tilde{a}_t = clip(\pi(s_t) + noise, a_{LB}, a_{UB}), \quad (24)$$

where a_{LB} and a_{UB} are the lower bound and upper bound of the action space, respectively; the function, $clip$, clips $\pi(s_t)+noise$ according to the lower bound and upper bound.

D. Closed-loop stability of RL-controlled BWR systems

Global stability of nonlinear uncertain dynamical systems, such as the BWR, is critical for ensuring plant safety and reliable performance. Analytical tools like Lyapunov second method (e.g., see [7], [15]) provide a sufficient (and hence possibly conservative) condition for stability. In other words, ensuring Lyapunov-based global stability of the BWR system may compromise its dynamic performance. Such a globally stable system, of which the strongest case is globally exponential stability, may fail to yield superior performance of the RLC-based closed-loop systems in comparison to the existing BWR controllers that are designed by other established control synthesis techniques such as H_∞. A popular and feasible option is testing of local stability at several operating points of interest, which does not examine the stability at other (potentially unstable) points of operation; nevertheless, like simulation, this approach may be capable of only identifying some of the potential sources of instability.

The above discussion evinces the fact that stability analysis of RLC-based BWR systems is indeed a very challenging task that needs a thorough analytical and experimental investigation. This issue of stability analysis of RLC-based BWR systems is suggested as a topic of future research in Section IV.

E. The pseudo-code of Deep Deterministic Policy Gradient

Based on the principles of *Deep Deterministic Policy Gradient* (DDPG) discussed in the Subsections III-B and III-C, the DDPG algorithm is summarized as a pseudo-code presented in Algorithm 1. It is noted that the Adam optimizer [25], instead of the gradient descent and gradient ascent, is used in this paper; however, in the pseudo-code, the optimizer gradient descent and gradient ascent are mentioned for the purpose of illustration only. The policy evaluation is made according to Algorithm 2, which is used only for convergence monitoring in the traning process, and it is not involved in the policy iteration process.

In the pseudo-codes, at time step t, when a_t is sent to the environment, the environment returns (s_{t+1}, r_{t+1}) after an internal state transition. The agent action (*i.e.*, control command), a_t, is the desired position of the control rod (*i.e.*, u in Eq. (9)). The environment internal state transition is conducted via Runge-Kutta integration of the reactor model depicted by Eqs. (1) to (4) and Eqs. (8) to (10). The step size of the Runge-Kutta integration is 0.001 sec and 10 steps of integration are conducted per state transition. Thus, the interval of a state transition is 0.01 sec. After the integration is complete, the updated & observable states are calculated and are returned as (s_{t+1}, r_{t+1}). The synthesis procedure of the observable states and the return function is presented in the next two subsections.

F. Identification of observable states

An environment needs to be created for the agent to interact with. To this end, three basic components in the environment need to be identified: (i) the environment dynamics; (ii) the observable states; and (iii) the reward function. The identification of the environment dynamics is straight forward in this paper, which is created by combining the dynamical systems of the BWR, NPMS and CRDS presented in Section II. The remaining tasks are to identify the observable states and to construct the reward function so that the agent can receive the state and the reward during interactions with the environment. In this subsection, the observable states are identified, while the reward function is constructed in the next subsection.

In the setting of BWR control, if only the measured excess neutron density, y, from neutronic power measurement system is used as an observed state for the RL agent, then the current operational condition of the reactor may not be adequately represented. For example, two different transition scenarios could have the same y_t value at time step t, and therefore the same action could be taken at time step t; however, at time step $t+1$, the next state, s_{t+1}, and the next reward, r_{t+1}, in those two scenarios could be totally different. This is due to the fact that the internal states of the environment for these two scenarios are not necessarily the same, even though the observed states are the same (*e.g.*, T_es are different but ys are the same). This kind of *Markov decision process* (MDP) problems is called *partially observable Markov decision process* (POMDP). To resolve the inherent difficulties of state representation in POMDP, various methods have been devised. Examples are: (i) usage of the recurrent neural network (RNN) [26]; or (ii) simply having

Algorithm 1: Deep Deterministic Policy Gradient (DDPG)

Randomly initialize policy and value neural networks parameters, $\boldsymbol{\theta}, \boldsymbol{w}$;
Create target policy and target value neural networks and conduct the parameters hard copy:
$\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$; $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$;
Create empty replay buffer R;
for episode $\leftarrow 1$ **to** M **do**
  Initialize environment with random set-point n^{ref} & system parameter μ and observe initial state of the environment, s_1;
  for $t \leftarrow 1$ **to** T **do**
    Select action: $a_t = clip(\pi(s_t|\boldsymbol{\theta}) + noise, a_{LB}, a_{UB})$;
    Send a_t to environment, and receive (s_{t+1}, r_{t+1});
    Store the quad $(s_t, a_t, r_{t+1}, s_{t+1})$ in R;
    Randomly sample a minibatch of K quads in R: $B = \{(s, a, r, s')_j\}_{j=1}^{K}$;
    Calculate TD error for each selected quad: $\delta = q(s, a; \boldsymbol{w}) - (r + \gamma q(s', \pi(s'; \boldsymbol{\theta}^-); \boldsymbol{w}^-))$;
    Update $\boldsymbol{w}$ by minimizing the TD errors (*e.g.*, using one-step gradient descent): $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha_1 \nabla_{\boldsymbol{w}} \frac{1}{|B|} \sum_{(s,a,r,s') \in B} \delta^2$;
    Update $\boldsymbol{\theta}$ by maximizing the value network output (*e.g.*, using one-step gradient ascent): $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_2 \nabla_{\boldsymbol{\theta}} \frac{1}{|B|} \sum_{s \in B} q(s, \pi(s; \boldsymbol{\theta}); \boldsymbol{w})$;
    Update $\boldsymbol{w}^-$ and $\boldsymbol{\theta}^-$: $\boldsymbol{w}^- \leftarrow \tau \boldsymbol{w} + (1-\tau)\boldsymbol{w}^-$, $\boldsymbol{\theta}^- \leftarrow \tau \boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta}^-$
  end
  Conduct policy ($\boldsymbol{\theta}$) evaluation according to Algorithm 2 for training convergence monitoring
end

Algorithm 2: Policy evaluation

Create empty list Rts for recording returns;
for test $\leftarrow 1$ **to** L **do**
  Initialize the return $Rt \leftarrow 0$;
  Initialize environment with random set-point n^{ref} & system parameter μ and observe initial state of the environment, s_1;
  for $t \leftarrow 1$ **to** T **do**
    Select action: $a_t = clip(\pi(s_t|\boldsymbol{\theta}), a_{LB}, a_{UB})$;
    Send a_t to environment, and receive (s_{t+1}, r_{t+1});
    $Rt \leftarrow Rt + r_{t+1}$
  end
  Store Rt in Rts;
end
Calculate average return: $Rt_{ave} \leftarrow \frac{\text{sum}(Rts)}{L}$

a vector of consecutive time steps of observed states (*e.g.*, $(y_t, y_{t-1}, \cdots, y_{t-k+1})$, where k is the length of the vector) to enhance the information provided to the agent to make decisions [27]. The first example needs longer time to train the recurrent layers; and the hyper-parameters of the recurrent neural network need to be tuned carefully, and this is computationally expensive. For the second example, a large number of consecutive observed states are required to circumvent the problem of under-representation of the environment internal states, which not only increases the number of parameters in the neural network, but also requires careful tuning of the length of the vector of the consecutive observed states. In this paper, the observable states at time step t are selected to be a 9-element vector s that does not involve recurrent layers and use only short-length consecutive observed states. The vector $s = [s1, s2, s3, s4, s5, s6, s7, s8, s9]$ consists of:

- $s1 = n^{ref}$, reference value (set-point);
- $s2 = y_t$, measured value at time step t;
- $s3 = a_{t-1}$, action at time step $t-1$;
- $s4 = n^{ref} - y_t$, difference between the reference value and the measured value at time step t;
- $s5 = \sum_{i=0}^{t}(n^{ref} - y_t)$, accumulated error from beginning to time step t, which is analogous to the integral;
- $s6 = y_t - y_{t-1}$, first-order difference, which is analogous to the derivative;
- $s7 = y_t - 2y_{t-1} + y_{t-2}$, second-order difference, which is analogous to the second derivative;
- $s8 = y_t - y_{t-2}$, difference between measured value at time step t and measured value at time step $t-2$;
- $s9 = y_t - y_{t-3}$, difference between measured value at time step t and measured value at time step $t-3$.

The selection of this 9-element vector is intuitive and the rationale is explained as follows. The component $s1$ is a specification of the target, where the system should be stabilized. The component $s2$ is the measured value from the neutronic power measurement system. Thus, via $s1$ and $s2$, the agent is informed of the reactor power target and the current measured power that carry the key information for controlling the reactor. The component $s3$ is the control command sent by the agent at the last time step, which can help the agent to generalize good actions on the cause-and-effect basis. The components $s4, s5$, and $s6$ correspond to the elements of proportional, integral, and derivative terms in a PID controller; even if the policy is fitted into a static linear model of $s4, s5, s6$, the RL controller should be tuned as a (sub)optimal PID controller. The component $s7$ provides augmented information on the trend of the change of the measurement. The components $s8$, and $s9$ (along with $s4$) inform the agent of the recent three steps of the trajectory in the form of relative values. These relative values are widely used in deep learning that can make the data stationary, which reduces the burden of generalization. The number of steps are chosen as three by trial-and-error through checking the convergence of the training. Compared to the training of recurrent layers for generalizing the observed states, the 9-element vector can be obtained directly without training. In contrast to only using a long vector of the measured values, the PID elements used for observation of states provide a more concise summary of the entire trajectory, and the second-order derivative provides additional information on the instantaneous rate of the measurement change.

G. Identification of the reward function

The reward function in the environment signifies what are the favored transitions. The reward can be designed by (1) returning a positive value if the action and state are favored and the larger the more favored, or (2) the reward can be a negative value to penalize the *not favored* actions and states; in this case, when the absolute value of the reward is larger, it means it is less favored, or (3) the reward can be both positive or negative. Whether the action and state are favored or not may depend on the control purpose. For example, if the control purpose is to drive a plant to reach a new operation state, the reward function could be the sum of (a) negative of the distance between current state and desired state ($-|$target state$-$current state$|$), and (b) negative of the absolute action ($-|$action$|$), because less actuation could be favored in the sense of saving energy or alleviate the burden on the actuator.

In this paper, the control objective is to stabilize the system to its power set-point. The reward used here has negative values because the objective is to penalize the deviation from the set-point. The reward in each step is set as negative of the ℓ_1-norm of the difference between the n and n^{ref},

$$R^d_{t+1} = -|n_{t+1} - n^{ref}|. \tag{25}$$

By maximizing the cumulative reward of R^d, an optimal policy should be able to stabilize the system at the set-point n^{ref} as fast as possible.

To minimize the mechanical wear of the control rod mechanism, the fast movement of the control rod is penalized:

$$R^a_{t+1} = -|\dot{z}_t|, \tag{26}$$

where $\dot{z} \triangleq \frac{dz}{dt}$ is calculated in Eq. (9). It is noted that $\dot{z}_t$ and $\dot{z}(t)$ are different; the former is evaluated at time step t where t means time step, while the latter is evaluated as a time derivative at the instant t in seconds.

It is also useful to constrain the internal state of the environment within a reasonable range to avoid unnecessary exploration by the agent. A constraint penalty is used to limit n in the range of (-1, 1). According the definition of n, when $n = -1$, the reactor has zero power; and when $n = 1$, the power of the reactor is doubled from its reference power value. The constraint penalty is

$$R^c_{t+1} = \begin{cases} 0 & \text{if } n_{t+1} \in (-1, 1), \\ W & \text{otherwise,} \end{cases} \tag{27}$$

where the value of W is manually tuned based on the convergence of the training. The total reward is a linear combination of these three rewards:

$$R_{t+1} = w_d R^d_{t+1} + w_a R^a_{t+1} + R^c_{t+1}. \tag{28}$$

The ratio w_d/w_a can be adjusted. When the ratio is higher, the control system more rapidly settles down to the set-point, and

the movement of the control rod would be aggressive, and vice versa. In this paper, the weights are chosen as $w_d = 10$ and $w_a = 0.01$ by trial-and-error to achieve rapid settling down to the set-point.

H. Actor and critic architectures

Figure 2 displays the neural network architectures of the actor and the critic, where both of them are built on the concept of multilayer perceptron [28] which is one of the most commonly used architectures for constructing neural networks. The critic neural network has two branches, one for the action and another for the observed states, and then they are concatenated together at later stages to predict the value defined in Eq. (16). The actor neural network has only one branch for generating the action. The last activation function before the action is a *tanh* function that outputs a value between -1 and 1. In the environment, it is scaled to -0.02 to 0.02 which makes the control rod saturation accountable. The selection of the number of neurons and the number of layers is based on trial-and-error that the selected architecture might not be optimal. To this end, a topic of future research is suggested in Section IV so that the neural networks can be further optimized for fast training and better policy generalization.

I. Results of the trained policy

The RL controller is trained by Algorithm 1 and after training, the parameters of policy network are kept constant for subsequent applications. The programming language is python and the neural networks are constructed and their parameters are updated by using PyTorch package [29] of python. The training clock time running on a Linux machine is approximately 45 s per episode on a PC with an Intel I9-10850K CPU and a Nvidia-1060-6Gb GPU.

The hyper-parameters of the agent and the environment are summarized in Table II. Variations of the set-point (n^{ref}) are restricted in the range of -5% to 5% around the reference steady-state value ($n|_{ss} = 0$). It is noted, however, that this range can be made wider at the expense of longer training time. The parameter μ is restricted in the range of $[0.5, 2.0]$, which consists of operations around fixed points, limit-cycles, and chaotic regions, as described in Subsection II-A. In each episode, the set point, n^{ref}, and the system parameter, μ, are randomly assigned within their respective ranges. The upper bound and lower bound of the action space are determined by the above ranges of set-point n^{ref} and parameter μ with the following reasoning. From Eqs. (1) to (4) and Eqs. (8) to (10), it can be found that, at steady-state equilibrium points, the control rod position should be $-\frac{\Lambda n^{ref}}{k_m}\left(\frac{\kappa_1 D}{\Lambda\kappa_2} + \frac{\kappa_1\mu\xi_0}{\kappa_2\kappa_4\Lambda}\right)$ which has the range of [-0.0151, 0.0151]. To provide additional flexibility of action selection, the lower bound and upper bound of the action space are extended in the range of [-0.02, 0.02].

To evaluate the convergence during training, the controller has been tested by 20 randomly generated environments (with different values of n^{ref} and μ) after each episode of training. The evolution of the training process is illustrated in Figure 3, where the abscissa is the number of episodes and the ordinate is $log(-Return)$.

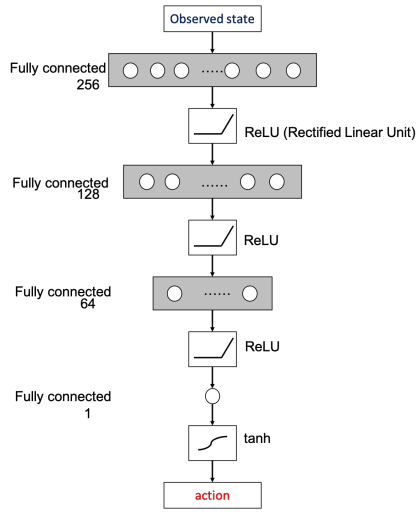
TABLE II: Hyper-parameters for agent and environment

Parameters	**Values**
Learning rate α_1 and α_2	1×10^{-4}
Discount rate γ	0.99
Soft update ratio τ	0.001
Minibatch size K	$1,024$
Number of episodes M	$3,900$
Replay buffer size	1×10^6
Episode horizon (time simulated per episode)	15 s
Step size of integration using RK4	0.001 s
Agent-environment interaction step size dt	0.01 s
Action space bounds in the agent (a_{LB}, a_{UP})	-1, 1
Action Gaussian noise standard deviation σ	0.01
Action scaling factor in the environment	0.02
μ range in the environment	0.5 to 2.0
Set-point (n^{ref}) range in the environment	-5% to 5%
Number of tests per policy evaluation L	20
Number of interactions per episode T	$1,500$
Constraint penalty W	$-2,000$
Reward weight w_a	0.01
Reward weight w_d	10

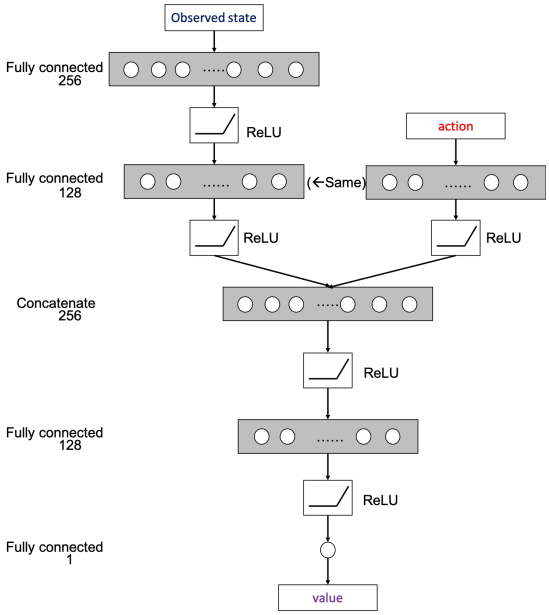
During training, after every 100 episodes, an average value of $log(-Return)$ of the tests is calculated during the immediately preceding 100 episodes, where there are a total of 2,000 tests = 100 episodes $\times$ 20 tests/episode; and the profile of the average over 2,000 tests is shown in Figure 4. For first 1000 episodes of the training, no convergence criterion is applied. After the first 1000 episodes, a convergence criterion is applied for training termination as follows. When the average of *Return* of the 2,000 tests does not increase (or equivalently the average of $log(-Return)$ of the 2,000 tests does not decrease), the training is said to have converged. The training lasts for $2,900$ episodes until convergence by the criterion (*i.e.*, the average of $log(-Return)$ at the 30th step is not less than that of the 29th step, as seen in Figure 4).

It is noted that the above convergence criterion does not guarantee an optimal policy; therefore, a sub-optimal policy is obtained with a limited number of episodes. After the training converges, 1,000 more episodes of training are used to confirm the convergence by manual inspection. The visualization of this convergence, presented in Figure 4, assures that a decreasing trend of the average value of $log(-Return)$ indeed ends after the 29th step. Increasing the training episodes may lead to deliver better policies at the expense of increased computation. After the training, the policy at 2,900 episodes is used as the delivered policy and its performance and stability are tested via the following simulations, where performance of the RL reactivity controller is reported for comparison with that of a $\mathcal{H}_\infty$ controller [3].

Figure 5 presents the performance comparison of the RL and $\mathcal{H}_\infty$ control systems for 1% step change in the power set-point, where the system parameter is selected at three different values as $\mu \in \{0.8,\ 1.5,\ 2.0\}$. It is shown that the BWR does not have larger power oscillations for all three cases when the RL controller is used. In contrast, significant power oscillations appear for $\mathcal{H}_\infty$ control at $\mu \in \{1.5, 2.0\}$, as seen in Figure 5. The step-response characteristics of the rise time, settling time, settling minimum, settling maximum



(a) Actor architecture



(b) Critic architecture

Fig. 2: Architectures of the actor (*i.e.*. policy neural network) and the critic (*i.e.*, value neural network)

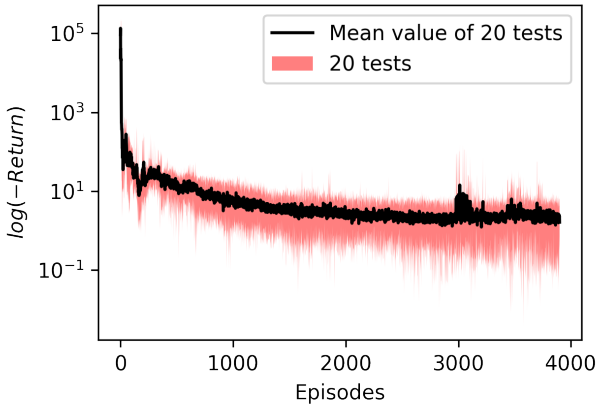


Fig. 3: Return evaluation during training process using 20 tests. The black line indicates the mean value of the 20 tests which are bounded in the red shaded region.

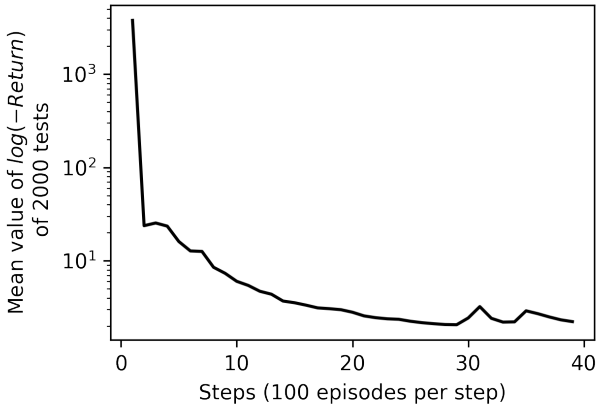


Fig. 4: Convergence inspection using 2,000 tests per step

and percentage overshoot are summarized in the Table III. Therefore, the RL controller appears to outperform the $\mathcal{H}_\infty$ controller in the nonlinear regions where $\mu = 1.5,\ 2.0$ in terms of all the above characteristics. In the linear region (*e.g.*, $\mu = 0.8$), RL control has shorter rise time & settling time and smaller overshoot while the settling minimum of the $\mathcal{H}_\infty$ control is slightly larger than that of RL-based control.

Figure 6 compares the disturbance rejection performance of the control systems. In this case, a step reactivity disturbance of 5 cents is inserted at the beginning of the simulation. The RL control outperforms the $\mathcal{H}_\infty$ control in all cases (i.e., $\mu \in \{0.8,\ 1.5,\ 2.0\}$) and it only takes around 0.16 s for the RL to settle down, while $\mathcal{H}_\infty$ needs several seconds for power settling, as seen in Table IV.

Simulation results in both Figure 5 and Figure 6 show that the RL control is capable of stabilizing the BWR. These results also show that RL control is less sensitive to system parameter perturbations as explained below.

In the simulations of power set-point step change, the settling time of the BWR using RL control retains around 0.38s as μ is changed from 0.8 to 2.0; in contrast, the settling time for $\mathcal{H}_\infty$ control changes from $\sim$ 0.41s to $\sim$ 9.7s. For reactivity disturbances, the settling time for RL control does not have any noticeable changes for various perturbations of the system parameter μ; in contrast, the settling time for $\mathcal{H}_\infty$ control would vary between 6.06s and 12.14s, as seen in Figure 6. These comparisons show that RL control is apparently more robust than $\mathcal{H}_\infty$ control.

The above simulations are standard ways to assess the performance of a control system. However, in actual operation of nuclear power plants (NPPs), the system parameter μ may not remain constant; it is also interesting to investigate the response under transient operations with different system parameter perturbations. Figure 7 qualitatively compares the per-

formance of RL control and $\mathcal{H}_\infty$ control under perturbations injected in various ways: step perturbation injections during both power transient (caused by step power change and power increase ramping) and constant power operations as well as sinusoidal perturbation injection at constant power operations. It is seen that the system using $\mathcal{H}_\infty$ control has obvious power oscillations while the system using RL control does not suffer from power oscillations. In these simulations, the RL control is seen to withstand various system perturbations (*i.e.*, high-frequency perturbations like step change and low-frequency perturbations like sinusoidal excitation), while the $\mathcal{H}_\infty$ control appears to be more tolerant of low-frequency perturbations than high-frequency perturbations.

TABLE III: Characteristics of step response of power change

Chr*	$\mu = 0.8$		$\mu = 1.5$		$\mu = 2.0$	
	RL	$\mathcal{H}_\infty$	RL	$\mathcal{H}_\infty$	RL	$\mathcal{H}_\infty$
RT	**0.0151**	0.0982	**0.0151**	0.0983	**0.0151**	0.0983
ST	**0.3870**	0.4078	**0.3819**	10.0763	**0.3783**	9.6719
SMin	0.0085	**0.0090**	**0.0085**	0.0081	**0.0085**	0.0070
SMax	**0.0104**	0.0120	**0.0104**	0.0119	**0.0104**	0.0119
OS	**3.9**%	19.7%	**3.7**%	19.1%	**3.6**%	18.7%

* Characteristics:

- RT (rise time [s]): time spent for signal to rise from 10% to 90 % of the desired value
- ST (settling time [s]): time spent for signal to settle within 2% absolute error between the signal and set-point
- SMin (settling min): minimum value of the signal after the signal has risen
- SMax (settling max): maximum value of the signal after the signal has risen
- OS (percentage overshoot): percentage amount that the signal overshoots the set-point

TABLE IV: Settling time (ST) for step change of reactivity

Chr*	$\mu = 0.8$		$\mu = 1.5$		$\mu = 2.0$	
	RL	$\mathcal{H}_\infty$	RL	$\mathcal{H}_\infty$	RL	$\mathcal{H}_\infty$
ST	**0.15**	12.14	**0.16**	6.64	**0.16**	6.06

* Characteristics:

- ST is the time spent for a signal to settle within 0.02% absolute error between the signal and the set-point after a disturbance occurs.

IV. Summary, Conclusions, and Future Research

This paper has synthesized the (nonlinear) reactivity control system for a BWR plant by making use of an artificial intelligence (AI)-based reinforcement learning (RL) algorithm, called *Deep Deterministic Policy Gradient* (DDPG). Performance of RL control has been compared with that of $\mathcal{H}_\infty$ control in terms of the rise time, settling time, settling minimum, settling maximum, and percentage overshoot in both nonlinear and linear regions of operation in the BWR model. Robustness of RL control is demonstrated (by simulation); the performance of the RL control system changes significantly less than that of the $\mathcal{H}_\infty$ control system under parameter perturbations and exogenous disturbances, and the stability of RL control is seen to be superior to that of $\mathcal{H}_\infty$ control in all cases. In general, it is observed that RL control outperforms $\mathcal{H}_\infty$ control in most of the (simulated) test cases.

It is noted that, in this paper, the stability of the BWR system is not established by rigorous mathematics (*e.g.*, Lyapunov

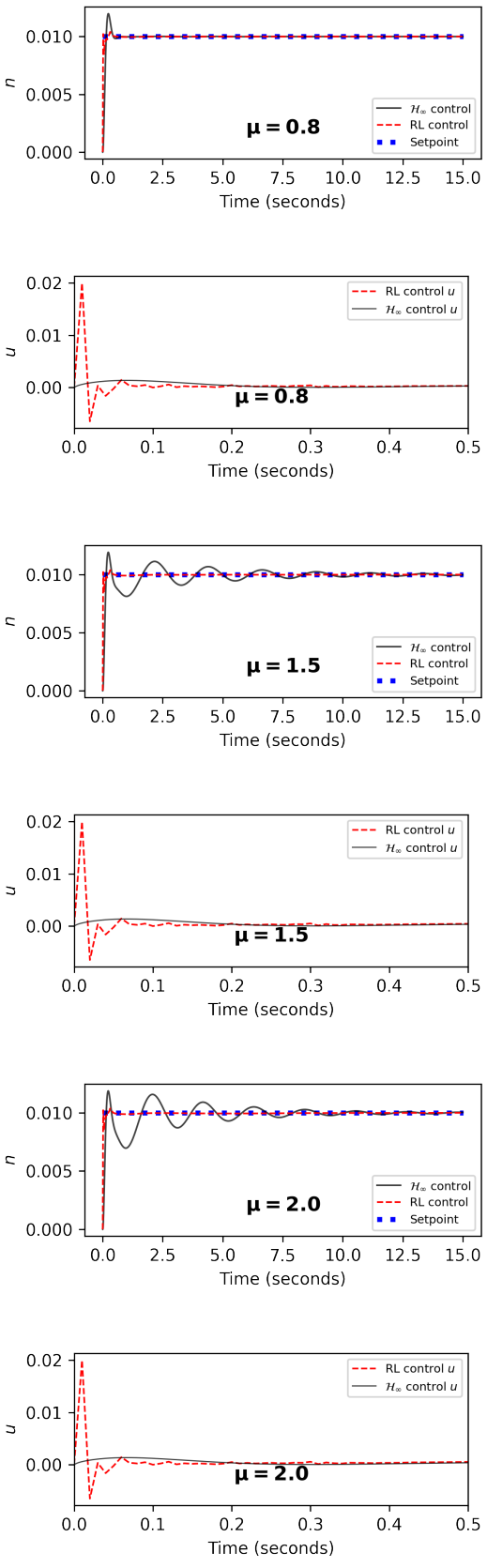


Fig. 5: Step response of 1% set-point change

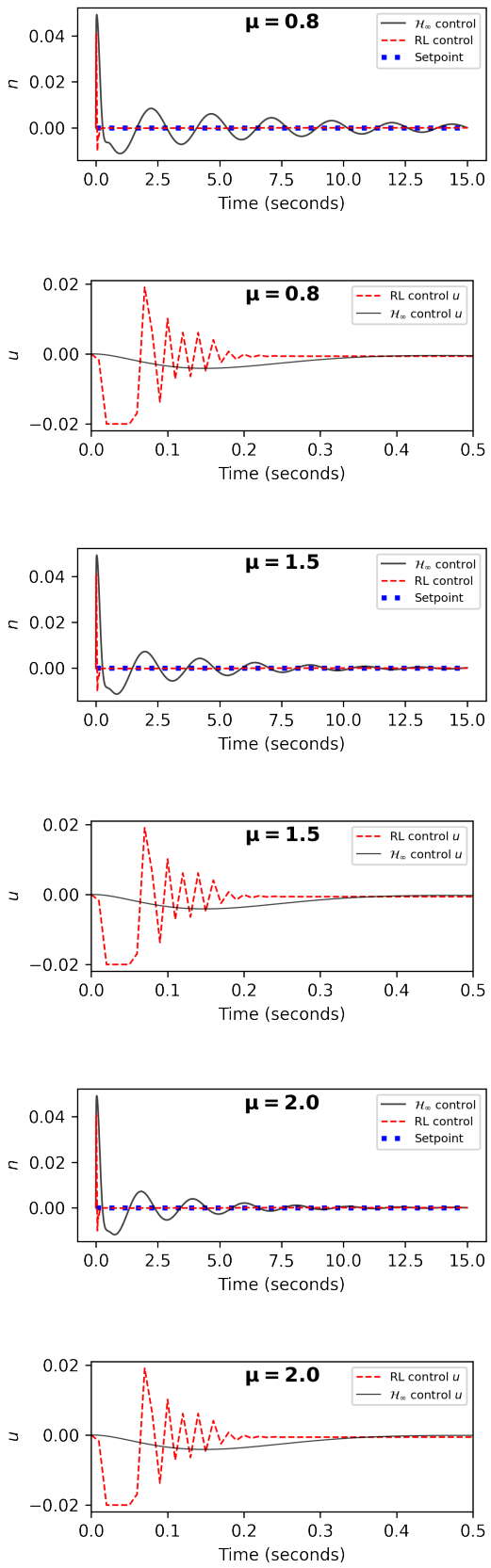


Fig. 6: Step response of 5 cents reactivity insertion

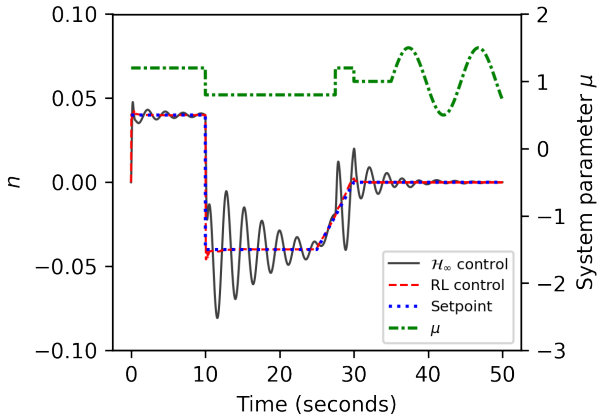


Fig. 7: Comparison of RL and $\mathcal{H}_\infty$ control systems for step and ramp power maneuver with system perturbations

stability conditions). However, the empirical results, derived from simulation, are encouraging, and these results should draw attention of the nuclear engineering community to enhance the technology of reactor control by further research in AI and RL. While there are many experimental and theoretical areas for future research, the authors suggest following topics:

1) *Optimal supervisory control*: An example is to optimize the plant performance by AI and RL. Another example is optimal set-point generation for local PID controllers.
2) *Tuning and sensitivity analysis of the hyperparameters*: The hyperparameters (*i.e.*, those parameters in Table II and Figure 2) should be tuned to enhance the training performance (*e.g.*, speed and stability of the training convergence) for the individual reactor under control.
3) *Stability analysis of RLC-based BWR systems*: Tools like Lyapunov second method (e.g., see [7], [15]) provide sufficient conditions for stability. While globally exponential stability of RLC is desirable, such a control system may often severely compromise the system performance. A possible choice of analysis could be bounded input bounded output (BIBO) stability [7]; alternatively, a trade-off analysis between robust stability and robust performance could be investigated, similar to what is done in H_∞ control.
4) *Comparison of different integrated AI and RL concepts for control system synthesis in nuclear reactors*: Different types of RL algorithms and various AI methods can be investigated, because a specific type of reactor may need a certain type of AI and RL integration for its optimal performance.
5) *Experimental validation of different AI and RL control concepts*: A control policy trained by RL in a simulator/model can be applied for its validation on a real-life reactor. A laboratory-scale nuclear reactor could be an ideal site for concept validation before its experimental validation on a commercial-scale nuclear power plant. Uncertainties in such an experimental work could be modeled as a noise term so that statistical significance could be assigned to the results for comparison with other controllers.

ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers and the journal editors for providing valuable suggestions which have significantly improved the technical quality of the paper. The work reported in this paper has been supported in part by the U.S. Army Research Office (Grant No. W911NF-20-1-0226) and U.S. National Science Foundation (Grant no. CNS-1932130). Any opinions, findings, and conclusions in this paper are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

REFERENCES

[1] T. Van der Hagen, A. Stekelenburg, and D. Van Bragt, "Reactor experiments on type-i and type-ii bwr stability," *Nuclear Engineering and design*, vol. 200, no. 1-2, pp. 177–185, 2000.

[2] J. March-Leuba and J. Rey, "Coupled thermohydraulic-neutronic instabilities in boiling water nuclear reactors: a review of the state of the art," *Nuclear Engineering and Design*, vol. 145, no. 1-2, pp. 97–111, 1993.

[3] K. Suzuki, J. Shimazaki, and Y. Shinohara, "Application of $\mathcal{H}$-∞ control theory to power control of a nonlinear reactor model," *Nuclear science and engineering*, vol. 115, no. 2, pp. 142–151, 1993.

[4] A. Arakawa, K. Sekimizu, and S. Sumida, "Fuzzy logic control application for bwr recirculation flow control system," *Journal of Nuclear Science and Technology*, vol. 25, no. 3, pp. 263–273, 1988.

[5] C. Lin and S.-R. Chang, "Adaptive predictive control of a boiling water reactor," *Nuclear science and engineering*, vol. 107, no. 2, pp. 158–172, 1991.

[6] Y. P. Pane, S. P. Nageshrao, J. Kober, and R. Babuška, "Reinforcement learning based compensation methods for robot manipulators," *Engineering Applications of Artificial Intelligence*, vol. 78, pp. 236–247, 2019.

[7] H. Khalil, *Nonlinear Systems, 3rd ed.* Prentice Hall, Upper Saddle River, NJ, USA, 2002.

[8] R. Kamalapurkar, P. Walters, J. Rosenfeld, and W. Dixon, *Reinforcement learning for optimal feedback control*. Springer, 2018, pp. 26-27.

[9] L. Buşoniu, T. de Bruin, D. Tolić, J. Kober, and I. Palunko, "Reinforcement learning for control: Performance, stability, and deep approximators," *Annual Reviews in Control*, vol. 46, pp. 8–28, 2018.

[10] D. P. Bertsekas, *Reinforcement learning and optimal control*. Athena Scientific Belmont, MA, 2019, pp. 40-42.

[11] D. Lee, A. M. Arigi, and J. Kim, "Algorithm for autonomous power-increase operation using deep reinforcement learning and a rule-based system," *IEEE Access*, vol. 8, pp. 196 727–196 746, 2020.

[12] J. Park, T. Kim, S. Seong, and S. Koo, "Control automation in the heat-up mode of a nuclear power plant using reinforcement learning," *Progress in Nuclear Energy*, vol. 145, p. 104107, 2022.

[13] D. Lee and J. Kim, "Autonomous emergency operation of nuclear power plant using deep reinforcement learning," in *International Conference on Applied Human Factors and Ergonomics*. Springer, 2021, pp. 522–531.

[14] W. Zhong, M. Wang, Q. Wei, and J. Lu, "A new neuro-optimal nonlinear tracking control method via integral reinforcement learning with applications to nuclear systems," *Neurocomputing*, 2022.

[15] Z. Dong, X. Huang, Y. Dong, and Z. Zhang, "Multilayer perception based reinforcement learning supervisory control of energy systems with application to a nuclear steam supply system," *Applied Energy*, vol. 259, p. 114193, 2020.

[16] M. I. Radaideh, I. Wolverton, J. Joseph, J. J. Tusar, U. Otgonbaatar, N. Roy, B. Forget, and K. Shirvan, "Physics-informed reinforcement learning optimization of nuclear assembly design," *Nuclear Engineering and Design*, vol. 372, p. 110966, 2021.

[17] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015, date of access: 2022-02-12. [Online]. Available: https://arxiv.org/abs/1509.02971

[18] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International conference on machine learning*. PMLR, 2014, pp. 387–395.

[19] J. March-Leuba, "Nonlinear dynamics and chaos in boiling water reactors," in *Noise and Nonlinear Phenomena in Nuclear Systems*. Springer, 1989, pp. 371–385.

[20] ——, "Dynamic behavior of boiling water reactors," Ph.D. dissertation, The University of Tennessee, 1984, p. 137.

[21] R. C. Hilborn *et al.*, *Chaos and nonlinear dynamics: an introduction for scientists and engineers*. Oxford University Press on Demand, 2000, pp. 11-17.

[22] H.-O. Peitgen, H. Jürgens, D. Saupe, and M. J. Feigenbaum, *Chaos and fractals: new frontiers of science*. Springer, 2004, vol. 106, pp. 560-604.

[23] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[24] M. v. Otterlo and M. Wiering, "Reinforcement learning and markov decision processes," in *Reinforcement learning*. Springer, 2012, pp. 3–42.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014, date of access: 2022-02-12. [Online]. Available: https://arxiv.org/abs/1412.6980

[26] L. T. Dung, T. Komeda, and M. Takagi, "Reinforcement learning for pomdp using state classification," *Applied Artificial Intelligence*, vol. 22, no. 7-8, pp. 761–779, 2008.

[27] J.-L. Kang, S. Mirzaei, and J.-A. Zhou, "Robust control and training risk reduction for boiler level control using two-stage training deep deterministic policy gradient," *Journal of the Taiwan Institute of Chemical Engineers*, 2021.

[28] J. Shin, T. A. Badgwell, K.-H. Liu, and J. H. Lee, "Reinforcement learning–overview of recent progress and implications for process control," *Computers & Chemical Engineering*, vol. 127, pp. 282–294, 2019.

[29] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.