Demonstration of Microreactor Reactivity Control with Reinforcement Learning

Conference Paper · April 2024

DOI: 10.13182/PHYSOR24-43480

4 authors, including:

Majdi I. Radaideh
University of Michigan
92 PUBLICATIONS **851** CITATIONS
SEE PROFILE

Dean Reid Price
Idaho National Laboratory
43 PUBLICATIONS **190** CITATIONS
SEE PROFILE

Kamal Abdulraheem
University of Michigan
16 PUBLICATIONS **81** CITATIONS
SEE PROFILE

Demonstration of Microreactor Reactivity Control with Reinforcement Learning

Majdi I. Radaideh [1,*], Dean Price[1], Kamal Abdulraheem[1], Moutaz Elias[2]

[1]Department of Nuclear Engineering and Radiological Science, University of Michigan, Ann Arbor, MI;
[2]Lam Research Corporation, Tualatin, Oregon, 97062

[leave space for DOI, which will be inserted by ANS]

ABSTRACT

As the reduction of operation and maintenance costs is a major goal for advanced reactors in general and microreactors in specific, the development of robust autonomous control algorithms is needed to ensure safe and autonomous reactor operation. Recently, artificial intelligence and machine learning algorithms, namely, reinforcement learning (RL) algorithms, have seen an exponential increase in their applications to control problems, including plasma control in fusion tokamaks and building energy management, among others. In this work, we introduce intelligent control employing RL in the context of nuclear microreactor control. The RL agent is trained through proximal policy optimization (PPO), a state-of-the-art deep RL approach, leveraging a high-fidelity simulation of a microreactor design motivated by the Westinghouse eVinci^{TM} design. We employed a Serpent model to generate data regarding drum positions, core criticality, and core power distribution for the training of a feedforward neural network surrogate model. Subsequently, this surrogate model is employed to guide a PPO control policy in determining the optimal drum position across various reactor states, ensuring critical core conditions and symmetrical power distribution across all six quadrants. The results demonstrate the outstanding performance of RL in the identification of optimal drum positions that can achieve a Quadrant Power Tilt Ratio (QPTR) value of approximately 1.002 (while adhering to the limit of $QPTR < 1.02$) and keeping criticality within a 10 pcm range. The results also show the ability of well-trained RL control policies to identify control actions at a high speed, which can be considered a promising approach to enable real-time autonomous control through digital twins.

Keywords: Reinforcement Learning, Reactor Control, Microreactors, Neural Networks

1. INTRODUCTION

One piece of enabling technology for the development of digital twins was identified by [1] to be methods for the automated and fast construction of reduced order models (ROM). In this context, a ROM is a fast-running model that predicts system parameters based on either sensor-obtained inputs or predicted quantities from other models. Furthermore, these ROMs should be constructed automatically without the need for operator intervention from data generated from the operating system. The primary challenge in autonomous control systems for nuclear reactors, particularly microreactors, arises from the need to balance computational efficiency with the precision of actions taken at specific reactor states. An optimal controller is characterized by its ability to deliver precise actions while minimizing computational overhead. This kind of controller

*radaideh@umich.edu

can be seamlessly incorporated into a digital twin, facilitating reactivity control in microreactors. This research delves into the potential of reinforcement learning as a means to create efficient and precise control policies for microreactors.

Reinforcement learning (RL) capabilities for optimization and control have been explored and proven. RL algorithms have seen an exponential increase in their applications to control problems, including plasma control in fusion tokamaks [2] and building energy management [3], among others. For nuclear reactor design optimization, which can be perceived as a static problem, several studies have demonstrated real-world applications of RL algorithms such as deep Q network (DQN) and proximal policy optimization (PPO) in assembly design optimization on small [4] and large scales [5] along with an open-source implementation framework [6]. In nuclear reactor control, the recent study by [7] has developed a nonlinear reactivity controller system for a boiling water reactor by making use of a RL algorithm called deep deterministic policy gradient (DDPG). The study has found that RL is more robust when compared to another control approach (H_∞) under parameter perturbations and exogenous disturbances. For non-nuclear power applications, plasma control in fusion Tokamak has been performed in this study [2] using maximum a posteriori policy optimization (MPO), which is an actor-critic algorithm variant of RL. The results have demonstrated RL ability to control a diverse set of plasma configurations on various Tokamak configurations, including elongated, conventional shapes, as well as other advanced configurations. Another study by [3] has explored hybridizing DQN with model predictive control (MPC) in order to provide an efficient management of building energy. The hybrid controller takes advantages of both RL learning capabilities and the MPC constraint adherence behavior.

This paper addresses a steady-state optimization and control problem for a heat pipe microreactor across three different burnup levels. Our approach utilizes proximal policy optimization to train a unified control strategy with the objective of pinpointing the ideal placement of the control drum, enabling a critical core and balanced power distribution in the core quadrants. The aim here is to establish a single control policy capable of identifying the optimal action across these three different reactor states. This investigation not only showcases the potential of RL in microreactor control but also illustrates the ability of the trained RL policy to execute control actions with minimal computational resources. The study is conducted on a microreactor design motivated by the Westinghouse eVinci^{TM} design and is simulated using Serpent. To accelerate RL training, the high-fidelity Monte Carlo model is used to develop a precise surrogate model for training the RL control policy.

2. METHODOLOGY

2.1. Heat Pipe Microreactor Model

In this study, Serpent [8] is used to calculate neutronic quantities associated with a heat pipe microreactor design motivated by the Westinghouse eVinci^{TM} design. The particular design used in this study will be referred to as the eVinci^{TM} motivated design (EMD) and is selected because its material and geometrical specifications are publicly-available in [9]. The EMD is designed to operate at 4 MWth with a 4 year lifetime, it has 12 control drums which are used to control reactivity. An illustration of the core is provided in Figure 1 where the Serpent-rendered core geometry at midplane is provided alongside a 3D rendering of the geometry. In this figure, the 12 control drums are shown at a 90° orientation in the 2D Serpent-rendered core geometry. These control drums can be rotated to manipulate the reactivity and power distribution of the core.

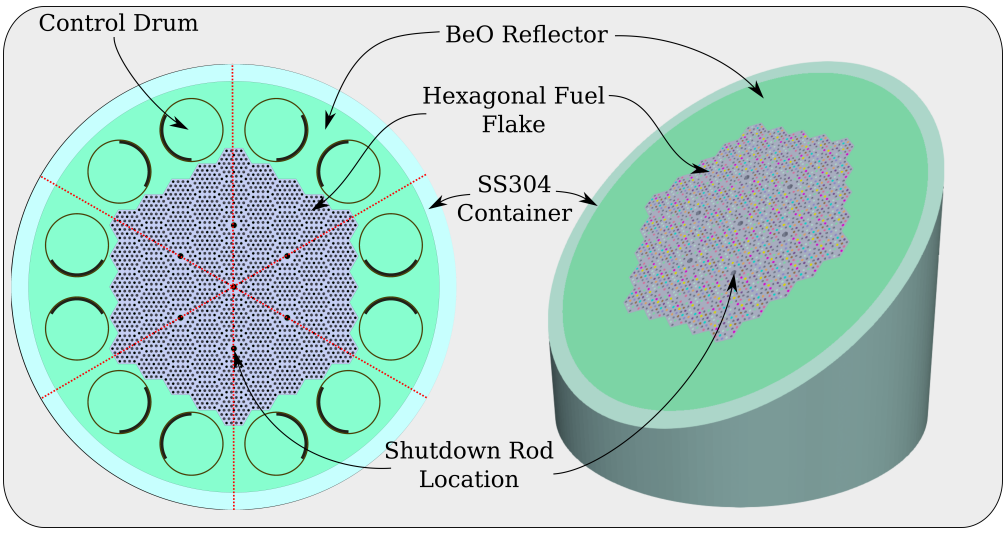

Figure 1. EMD with control drums oriented at 90°, shown with the slice at core midplane (left). Hexant divisions are shown with dotted red lines. In addition, a 3D depiction of the EMD with diagonal cut (right). Control drums are not rendered in the 3D depiction.

2.2. Surrogate Modeling

In this analysis, calculating neutronic core characteristics using the Serpent neutronics code would be too computationally expensive for the large number of evaluations required for the RL algorithm. Ideally, the control problem would be integrated into a digital twin which would be fed with real-time sensor data—there would be no need for any direct calculation of these quantities. Therefore, a set of surrogate models were created to predict core properties given a set of control drum positions at a particular cycle time. The relevant core properties for the control problems considered here are the core multiplicity and hexant power fractions. The hexant power fractions is a set of 6 quantities which describe the fraction of total power which comes from 6 portions of the core. The divisions used to calculate these hexant powers are shown in Figure 1 with red dotted lines. The control drums within each hexant are considered to move together, in opposite directions such that the positions of the 12 total control drums can be described with a set of 6 control drum angles.

A nominal core depletion calculation is run in Serpent with all control drums facing 90°. Then, three fuel composition states can be identified, 0YR corresponds to the beginning of life fuel composition when the core is loaded with fresh fuel, 2YR corresponds to the core fuel composition after 2 years of operation and 4YR corresponds to the core fuel composition after 4 years of operation. Serpent is used to calculate core multiplicities and hexant powers for 250 control drum positions for each of these 3 fuel compositions for a total of 750 calculations. The Monte Carlo sampling statistics were selected such that the final uncertainty in core multiplicity was estimated to be about 9 pcm. Within each fuel composition, symmetries in the core geometry can be exploited to turn these 250 calculations into 3000 data points which can be used to train and evaluate the surrogate models. These 3000 data points are divided into a training data set, used to calculate weights and biases associated with the DNN, a validation set, used to select an optimal network architecture and a testing set, used to evaluate the final performance of the DNN, with a 70%/15%/15% split, respectively. The divisions between these sets did not split data points which may have come from the same Serpent calculation (symmetries were applied after data is divided among the three data sets).

For each of the 0YR, 2YR and 4YR, a single DNN with 6 inputs and 7 outputs was created to simultaneously

predict core multiplicity and 6 fractional hexant powers given a set of 6 control drum angles. For architecture optimization, the number of hidden layers, nodes per hidden layer and learning rate are to minimize the mean squared error when predicting the validation data set independently for each of the 3 DNNs corresponding to the 0YR, 2YR and 4YR cases. A constant number of nodes per hidden layer is used to simplify the optimization process. Bayesian optimization is used for this mixed-integer optimization problem with an allowance of 30 training processes. The results from this training process, including performance measures of the surrogate models, are discussed in Section 3.1.

2.3. Reinforcement Learning (RL) and Control

Proximal Policy Optimization (PPO) [10] is a deep RL algorithm that aims to optimize a policy to maximize the expected cumulative reward in a Markov decision process (MDP). It belongs to the class of Policy Gradient (PG) methods and is designed to address some of the challenges associated with training deep RL agents. In the traditional PG approach, the policy is trained to map states to appropriate actions by optimizing the following loss function

$$L^{PG}(\theta) = \mathbb{E}[log\ \pi_\theta(a_t|s_t)\hat{A}_t], \tag{1}$$

where $\mathbb{E}$ is the expectation over a batch of time transitions (t), π is the policy to be optimized which has weights θ, and $\hat{A}_t$ is an estimate of the advantage function described below. The policy π predicts action a given state s at time step t. *Such policy can be trained in a similar fashion to optimal control policies, where the policy is supposed to take the most optimal action given the state of the system.*

PPO on the other hand operates by iteratively improving the policy while ensuring stable and efficient learning compared to PG learning which is often noisy and unstable. PPO employs a surrogate objective function to perform policy updates. This objective function encourages an increase in the probability of actions that have higher estimated advantage over others while constraining the policy update to prevent large changes. To ensure stable policy updates, PPO introduces a clipped surrogate objective $L^{clip}(\theta)$, which does not exist in PG learning and it encourages small policy changes. The clipped objective is defined as:

$$L^{clip}(\theta) = \mathbb{E}\left[\min\left(R_t^\pi(\theta)\hat{A}_t, \text{clip}\left(R_t^\pi(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right)\right], \tag{2}$$

where:

- $R_t^\pi(\theta)$ is the probability ratio, which is the probability of taking an action under the new policy π_θ divided by the probability under the old policy $\pi_{\theta_{\text{old}}}$.
- $\hat{A}_t$ is an estimate of the advantage function, indicating how much better an action is compared to the average action.
- ϵ is a hyperparameter controlling the degree of policy change (a.k.a clipping range).

In addition to the clipped objective, PPO includes a value function loss to estimate the value of states. The value function loss can be defined as:

$$L^V(\theta) = \mathbb{E}\left[\left(V_\phi(s_t) - V_t^{\text{target}}\right)^2\right], \tag{3}$$

where $L^V(\theta)$ is the value function loss, $V_\phi(s_t)$ is the estimated value of state s_t with parameter ϕ, and V_t^{target} is a target value for state s_t, often computed using the rewards and next state values.

The total PPO objective combines the clipped PPO objective in Eq.(2) with the value function loss in Eq.(3) along with an optional entropy term $\text{H}[\pi_\theta](s_t)$. This is the overall objective that PPO aims to optimize:

$$L^{\text{total}}(\theta) = L^{clip}(\theta) - c_1 L^V(\theta) + c_2 \text{H}[\pi_\theta](s_t), \tag{4}$$

where c_1 is a hyperparameter called value function coefficient, typically in the range of [0.5,1], c_2 is a hyperparameter called entropy coefficient, which controls the policy entropy, and it is typically in the range of [0,0.3], and H $[\pi_\theta]$ (s_t) is the entropy term in PPO, which is used as a regularization component in the objective function to encourage exploration by adding a level of uncertainty to the agent's policy. When $c_1 = 0.5$, the training will give equal weights toward optimizing the clipped loss as well as the value function loss. If $c_1 > 0.5$, the value function loss will be given more attention during training. The entropy of the policy measures the unpredictability or randomness of the agent's actions. By adding entropy, the agent is encouraged to explore a wider range of actions, which can be useful when the agent's policy is too deterministic. However, large entropy coefficient values can lead to a noisy policy that may fail to converge.

The PPO algorithm then seeks to optimize the policy parameter θ by maximizing the objective function $L^{total}(\theta)$ while maintaining policy stability and ensuring that policy updates (π_θ) are "proximal" to the old policy $(\pi_{\theta_{\text{old}}})$. The algorithm involves collecting samples from interactions with the environment, estimating advantages, and performing multiple iterations of policy updates to improve the policy over time.

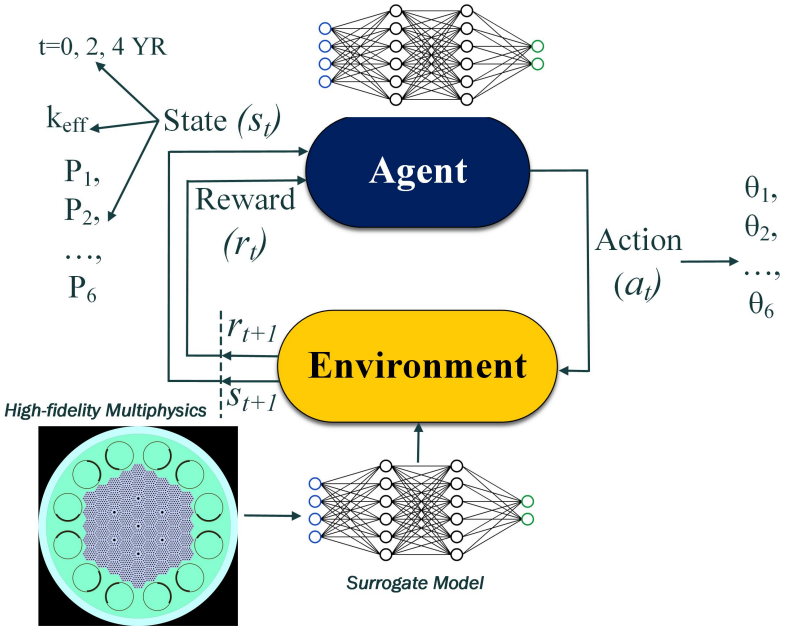

Figure 2. The RL control framework adopted in this work

Figure 2 illustrates the RL framework implemented with PPO, and for it to function effectively in our specific context, it necessitates the specification of several essential components, which can be described as follows:

- *Action space* (a_t): The action space in this problem represents the drum orientation (angle) that should be placed at during operation. The action space is discretized to 1° to simplify the search space. A multi-discrete object of size 6 with a range from 0° to 180° is used to represent the drum orientation in each quadrant. In future work, the problem can be extended to a continuous space. As the impact of the control drum angle on core properties generally follows with roughly the cosine of the control drum angle, the elimination of the control drum angles from -180° to 0° eliminates the possibility of multiple optimal solutions existing in the search space.
- *State space* (s_t): The state space in this problem consists of three parts: (1) the current time step (0 YR, 2 YR, 4 YR), (2) the core multiplicity (k_{eff}), and (3) the status of the power fraction of each quadrant $(P_1, P_2, ..., P_6)$. Typically, the state of the reactor dictates the action the agent should take to maximize the reward.

- *Environment*: The environment is where the agent interacts with the real world (e.g., in this case a simulation environment). The environment is represented by the surrogate model described in Section 2.2.
- *Reward (r_t)*: The reward function is simple and aims to ensure power symmetry in the six quandrants and a ctitical core at each time step t. The reward can be calculated as follows:

$$r_t^0 = \frac{1}{f_1^t + f_2^t}, \quad t = 0, 2, 4\ YR, \tag{5}$$

where

$$f_1^t = \frac{1}{6}\sum_{i=1}^{6} |P_i - 0.166667|, \tag{6}$$

$$f_2^t = |k_{eff}^t - 1.00000|. \tag{7}$$

Since RL is conventionally built to maximize rewards, the reciprocal is taken in Eq.(5) to convert a min to max problem without changing the sign. As an example, if the reactor is at $t = 0$ YR, with some random k_{eff} and P_i values, the RL agent should figure out the orientation of the six drums (a_t) to maximize r_t such that k_{eff} is as close to 1.0 as possible and all P_i's are as equal as possible. For monitoring purposes only to avoid tracking and plotting six P_i values, we define Quadrant Power Tilt Ratio (QPTR) as

$$QPTR = \frac{6 \times max(P_1, P_2, ..., P_6)}{\sum_{i=1}^{6} P_i}. \tag{8}$$

QPTR is a popular safety parameter where it must be maintained below ≤ 1.02 during operation.

- *Episode*: The episode defines the game the agent will be playing to maximize its reward. The episode always starts with the reactor at time $t = 0$ YR with random k_{eff} and P_i values, which represent the initial state s_1. The agent takes the action (a_1) and calculates the reward (r_1). The state is then updated to s_2, which is equivalent to $t = 1$ YR with new randomly sampled k_{eff} and P_i values. The agent then repeats the same process to get to s_3 at $t = 4$ YR. The goal is to maximize the reward of the episode by taking the optimal action in the three time steps. After the episode terminates, a new episode starts and the process is repeated many times until the agent learns how to play the game. Due to the sequential nature of the episode and as will be explained later in Section 3.2, we modify the definition of the reward function in Eq.(5) to:

$$\begin{aligned} r_1 &= r_1^0, \\ r_2 &= MEAN(r_1^0, r_2^0) - STD(r_1^0, r_2^0), \\ r_3 &= MEAN(r_1^0, r_2^0, r_3^0) - STD(r_1^0, r_2^0, r_3^0). \end{aligned} \tag{9}$$

We noticed that such a change in the reward function forces the agent to prioritize the learning of the three time steps simultaneously rather than getting biased and focused on increasing the reward for a single time step that might be easier to optimize, which would happen if the reward function in Eq.(5) is used with PPO. To get an excellent reward, the mean of the reward of the three reactor states must be as high as possible, while their standard deviation must be as low as possible.

The last important hyperparameter for PPO to mention is N_{steps}, which expresses how long the agent should interact with the environment before updating the policy. For example, $N_{steps} = 3$ will update the policy parameters after every episode (which could be too soon and lead to unstable and premature policy), while $N_{steps} = 300$ will update the policy after 100 episodes (which could be too late and may waste a lot of computing time through environment interactions). Finding a good balance between environment interactions and when to execute policy updates is a critical factor for PPO and RL training in general.

3. RESULTS

The results of this paper are presented in three subsections. First, the surrogate model of the Serpent simulation is trained and validated. The surrogate model is then used to train PPO policy in the second subsection. Lastly, the trained policy is tested to ensure that it can identify the optimal drum positions.

3.1. Surrogate Model Performance

In total, 3 different surrogate models for 0YM, 2YM and 4YM are used to supply core multiplicity and hexant powers (as fractions of total power) for a given set of control drum angles. In order to generate a network architecture which can adequately capture the relationships between control drum angles and core properties, a hyperparameter optimization routine is run where the learning rate, number of layers and nodes per layer are selected to ensure strong performance. After the DNNs are trained with these architectures, their performance is evaluated on a set of 456 testing data points. The performance metrics of these DNNs on this testing set, as well as their network architectures, are shown in Table I. For the hexant powers, the surrogate model produces 6 different values, each corresponding to a fractional power in the corresponding section of the core. For brevity, the performance metrics provided in Table I pertaining to these hexant powers have been averaged such that only a single set of performance metrics is provided. There is relatively little variation in model performance when predicting the powers in each hexant because the symmetry used to expand the training data.

Besides neural networks, we have trained other model types like linear regression and random forests. Linear regression provided a weak performance while random forests provided $R^2 \sim 0.8$. We continued with neural networks due to their better performance.

Table I. Network architecture and performance of surrogate models evaluated on testing set. Mean absolute error (MAE) and coefficient of determination (R^2) are shown when using the surrogate model to predict core multiplicity (k) and fractional hexant powers (P).

Parameter	**0 YR**	**2 YR**	**4 YR**
Learning Rate	3×10^{-3}	2×10^{-3}	2×10^{-4}
Num. Layers	5	5	7
Nodes per Layer	150	200	191
k**: MAE**	85 pcm	83 pcm	79 pcm
k**:** $\mathbf{R}^2$	0.996	0.996	0.998
P**: MAE**	2.32×10^{-4}	2.33×10^{-4}	2.29×10^{-4}
P**:** $\mathbf{R}^2$	0.999	0.999	0.999

3.2. RL Training Results

PPO was trained in parallel using 20 processors where each processor represents an agent. After experimenting with the hyperparameters of PPO through grid search, we found that the optimal set of hyperparameters is: $N_{steps} = 300$ for each agent (i.e., effectively the policy is updated every $20 \times 300 = 6000$ time steps), $c_1 = 0.75$, $c_2 = 0.01$, and $\epsilon = 0.4$. It can be noticed that this problem requires a long time horizon before policy update and relatively a large clipping range to achieve an optimal performance. In addition, providing more weight to the value function loss term through c_1 and the use of some policy entropy through c_2 also proved to be more effective. After increasing c_2 to values like 0.1 or 0.2, the reward values deteriorated and the training failed to converge to a good policy. We have utilized the PPO implementation of the open-source package Stable Baselines [11], where we have used the default architecture for the control policy, which is a 2-layer feedforward neural network with 64 nodes in each layer. *Note that the control policy and the*

surrogate model are two different neural network models as shown in Figure 2.

The results of PPO training for 1.2×10^6 time steps are shown in Figure 3 in three subplots showing how the core criticality, QPTR, and reward function are converging to optimal values. *It is important to notice that the results in Figure 3 are averaged over each episode and over the parallel processors, so they represent the average performance of the 20 parallel agents over the three time steps: 0 YR, 2 YR, 4 YR*. Some important observations from Figure 3 can be highlighted:

1. In order to offer more comprehensive monitoring, an epoch is defined by aggregating data from every **30,000** time steps (equivalent to 10,000 episodes). For instance, in a training regimen encompassing a total of 1.2×10^6 time steps, this results in a total of 40 epochs.
2. The learning trajectory of PPO is quite obvious, starting with agents exploring the environment by executing random actions, leading to low rewards, subcritical states, and elevated QPTR values during the initial training stages. However, as the agents progressively enhance their performance, the rewards begin to show rapid improvement, aligning with the convergence of k_{eff} and QPTR toward their target values.
3. The statistical metrics display a trend toward convergence after around **20 epochs**, illustrated by the narrowing of error bars and the alignment of minimum, maximum, and mean values. This alignment suggests that all agents have now developed the capacity to identify optimal actions to take, irrespective of the particular reactor state they encounter.

In addition, we noticed that the agent may get biased towards favoring the actions for a certain time step and forget other ones. For example, we realized that the agent masters the states that involve $t = 4$ YR quicker than $t = 0$ YR and $t = 2$ YR. This has led the reward to continue to increase due to continuous improvement of the solution at $t = 4$ YR, while the solutions at the other two time steps remain poor. Therefore, the change in the reward function in Eq.(9) that forces the agent to prioritize all time steps was essential to get robust training results.

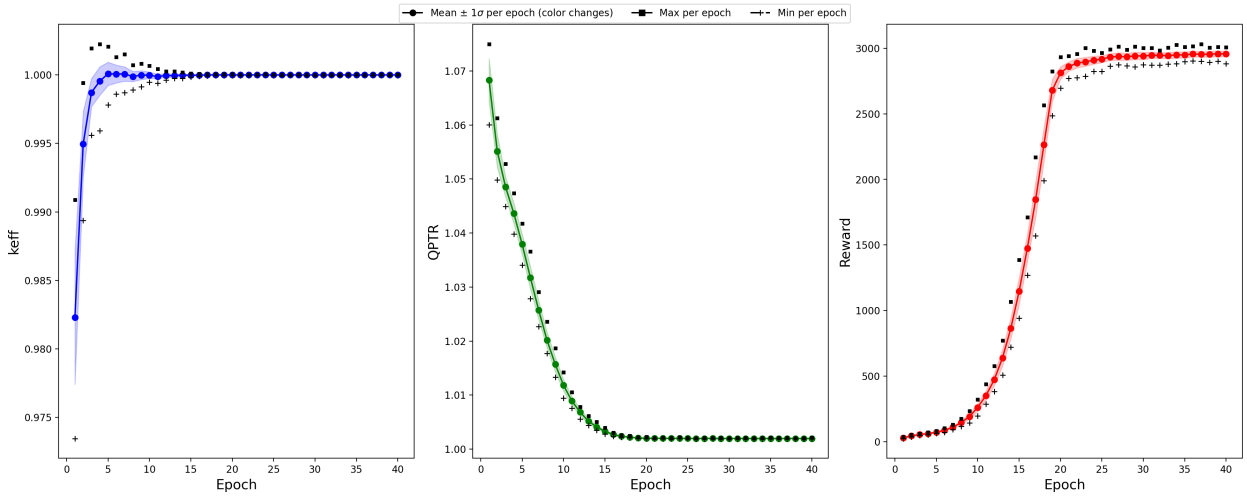

Figure 3. Progression of PPO training as a function of the number of epochs. Each epoch comprises 30,000 time steps for which the statistics (mean, std, max, min) are derived. The left figure shows the convergence of k_{eff}, the middle figure shows the convergence of QPTR, while the right figure shows the convergence of the reward value as indicated in Eq.(9). All values shown are averaged over the three reactor states and over the 20 parallel agents

3.3. RL Testing Results

Following the training of the PPO policy, we proceeded to evaluate its performance by utilizing its most recent version. We tested the policy by tasking it with predicting the optimal drum position for multiple random states at various time steps, with a short list of 10 states displayed in Table II. Evidently, the PPO policy exhibited a robust ability to make accurate predictions for the drum angle, regardless of the specific reactor state it encountered. This is evident from the excellent values of k_{eff} and QPTR achieved in these predictions. While the training of the PPO policy, as detailed in section 3.2, requires about 60 minutes to complete, the testing phase is considerably faster, as the policy can provide drum position predictions for each state in Table II in just around $\sim 0.025\ s$. This rapid prediction capability aligns with the primary objective of this study and underscores a major advantage of model-based control through RL to enable real-time autonomous control.

Although the results in Table II are reasonable, they can still be improved with a more refined hyperparameter search and more engineered reward functions. In this study, we focused on demonstrating that RL can still provide reliable performance without having to carry out a tedious hyperparameter search and with simple reward functions, which are the items we are considering in the extension of this work.

Table II. Testing results of the trained PPO policy for 10 different random states

Sample	Time (YR)	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	k_{eff}	QPTR
1	4	138	136	129	141	134	138	1.00000	1.002824
2	0	91	89	95	88	93	89	1.00002	1.002679
3	2	114	113	112	111	114	113	0.99993	1.001708
4	0	91	89	95	88	93	89	1.00002	1.002679
5	2	114	113	112	111	114	113	0.99993	1.001708
6	4	138	136	129	141	134	138	1.00000	1.002824
7	2	114	113	112	111	114	113	0.99993	1.001708
8	4	138	136	129	141	134	138	1.00000	1.002824
9	0	91	89	95	88	93	89	1.00002	1.002679
10	4	138	136	129	141	134	138	1.00000	1.002824

4. CONCLUSIONS

In this work, we have presented a preliminary application of intelligent control employing reinforcement learning (RL) to nuclear heat pipe microreactors. The RL agent is trained utilizing proximal policy optimization (PPO), a cutting-edge deep RL methodology, leveraging a high-fidelity simulation of a microreactor design motivated by the Westinghouse eVinci^{TM} design. A Serpent model was employed to produce high-fidelity data concerning drum positions, core criticality, and core power distribution for surrogate model training. Subsequently, the surrogate model is harnessed to instruct a PPO control policy to ascertain the crucial drum position across various reactor states, ensuring critical core conditions and symmetric power distribution within all six quadrants. The results illustrate the remarkable performance of RL in identifying optimal drum positions capable of achieving a QPTR value of approximately 1.002 (within the limit of $QPTR < 1.02$) and maintaining criticality within a range of 10 pcm. The computational cost of RL policy training was close to 60 min, while the policy control predictions in the deployment phase can take around $\sim 0.025\ s$. Future extensions of this work include expanding the scope to include additional control variables within the state space (e.g., temperature control), implementing advanced techniques for engineering reward functions to expedite convergence, and transitioning from discrete to continuous control problems. The research team anticipates that this study and its future extensions will unveil the potential of employing high-fidelity simulations for RL-based control policy training for digital twin applications.

ACKNOWLEDGEMENTS

The work in this paper was supported by various sources:

- Majdi I. Radaideh: Department of Energy, Office of Nuclear Energy Distinguished Early Career Program (Award: DE-NE0009424)
- Dean Price: Department of Energy, Office of Nuclear Energy, Integrated University Program Graduate Fellowship.
- Kamal Abdulraheem: Michigan Institute of Data Science, Eric and Wendy Schmidt AI in Science Postdoctoral Fellowship.

This research also made use of Idaho National Laboratory computing resources which are supported by the Office of Nuclear Energy of the U.S. Department of Energy under Contract No. DE-AC07-05ID14517.

REFERENCES

[1] B. Kochunas and X. Huan. "Digital twin concepts with uncertainty for nuclear power applications." *Energies*, **volume 14**(14), p. 4235 (2021).

[2] J. Degrave, F. Felici, J. Buchli, M. Neunert, B. Tracey, F. Carpanese, T. Ewalds, R. Hafner, A. Abdolmaleki, D. de Las Casas, et al. "Magnetic control of tokamak plasmas through deep reinforcement learning." *Nature*, **volume 602**(7897), pp. 414–419 (2022).

[3] J. Arroyo, C. Manna, F. Spiessens, and L. Helsen. "Reinforced model predictive control (RL-MPC) for building energy management." *Applied Energy*, **volume 309**, p. 118346 (2022).

[4] 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*, **volume 372**, p. 110966 (2021).

[5] M. I. Radaideh, B. Forget, and K. Shirvan. "Large-scale design optimisation of boiling water reactor bundles with neuroevolution." *Annals of Nuclear Energy*, **volume 160**, p. 108355 (2021).

[6] M. I. Radaideh, K. Du, P. Seurin, D. Seyler, X. Gu, H. Wang, and K. Shirvan. "NEORL: NeuroEvolution Optimization with Reinforcement Learning—Applications to carbon-free energy systems." *Nuclear Engineering and Design*, **volume 412**, p. 112423 (2023).

[7] X. Chen and A. Ray. "Deep reinforcement learning control of a boiling water reactor." *IEEE Transactions on Nuclear Science*, **volume 69**(8), pp. 1820–1832 (2022).

[8] J. Leppänen, M. Pusa, T. Viitanen, V. Valtavirta, and T. Kaltiaisenaho. "The Serpent Monte Carlo code: Status, development and applications in 2013." *Annals of Nuclear Energy*, **volume 82**, pp. 142–150 (2015).

[9] D. Price, N. Roskoff, M. I. Radaideh, and B. Kochunas. "Thermal Modeling of an eVinci™-like heat pipe microreactor using OpenFOAM." *Nuclear Engineering and Design*, **volume 415**, p. 112709 (2023).

[10] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms." *arXiv preprint arXiv:170706347* (2017).

[11] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. "Stable-baselines3: Reliable reinforcement learning implementations." *The Journal of Machine Learning Research*, **volume 22**(1), pp. 12348–12355 (2021).