Real-time Optimization of Industrial Processes using Deep Reinforcement Learning

Nitin Singh[1], Jasper Stolte[2], Stanislav Jaso[2], Bei Li[2], Reinier van de Pol[2], and Christian Michler[2]

[1]Shell India Markets Private Limited, Bangalore, India
[2]Shell Global Solutions International B.V., Amsterdam, The Netherlands

Abstract

Reinforcement learning (RL) has shown immense potential in various applications; however, its application in complex industrial processes is yet to be widely explored. This work aims to explore the potential of RL in process engineering and control through a proof of concept study to demonstrate the application of RL for real-time optimization (RTO) in a catalytic reactor system. The objective is to maximize the production of a high-value hydrocarbon while ensuring process constraints. A suitable actor-critic RL architecture is used, and the results are compared with a mathematical optimization solver-based benchmark. The study also evaluates the capabilities of Microsoft Project Bonsai, an AI platform for designing autonomous systems. Major contributions of this work include demonstrating the application of RL for RTO problems in chemical processes, discovering the adaptiveness and fast inference time of RL, and presenting a method for handling constraints during policy network training. The results show that RL can find feasible solutions comparable to the optimization-based benchmark.

1 Introduction

The field of Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) has made significant advances in recent years, yielding impressive results in diverse areas such as robotics, AI game playing agents, automated stock trading, self-driving cars etc. (Silver et al., 2016), (Silver et al., 2017), (Mnih et al., 2013), (Mnih et al., 2017), (Xiong et al., 2017). The outcomes have generated a growing interest in applying RL to further real-world engineering and physical systems where a dynamic control is required like Industrial Process Control. While a considerable amount of past research in this field has concentrated on RL to determine discrete-time optimal control policies for nonlinear processes (Spielberg et al., 2019), (Ernst et al., 2008), (Nian et al., 2020), there have been relatively few studies that have investigated RL's potential for real-time optimization (RTO) applications. In RTO, the primary function is to maximize system performance or a key metric, rather than maintaining set points (Powell et al., 2020). Through a proof of concept study, this work explores the application of RL for RTO in a catalytic reactor system.

The broader purpose for this project is to explore the application of RL in the process engineering and control domain, draw key conclusions and learnings, and assess its potential for delivering future value. Despite several promising results in other areas, RL is yet to be widely applied in complex industrial processes, primarily due to two main challenges: *(i)* Dealing with high-dimensional action and state spaces - a typical scenario in industrial processes - poses a complex challenge, and *(ii)* difficulty in satisfying *state constraints*[1] (Pan et al., 2021), (Nian et al., 2020), (Dalal et al., 2018).

Given these challenges, the set-point finding problem presents itself as a more suitable starting point for exploring the potential of RL as it can be considered a simpler representative version of the full dynamic set-point tracking problem. Additionally, for comparison purposes, we had access to a classical mathematical optimization based benchmark. It is, however, acknowledged that the set-point tracking problem is more suitable from both a methodological and application standpoint. While the application of RL for steady-state RTO is not ideally suited considering the fundamental principles of RL in Markov Decision Processes (MDP's) and system dynamics (Kaelbling et al., 1996), there is still an unexplored potential for gaining more insights for its use in RTO applications. Notwithstanding, we attempt to introduce (pseudo) system dynamics through the utilization of a method called here as *delta actions* (further explained in Section 4.2.1).

In this study, we present an RL-based approach to maximize the production of a certain high-value hydrocarbon in a steady-state catalytic reactor system while satisfying process constraints and ensuring that input and output process variables remain within their prescribed bounds. In other words, we solve a complex, constrained non-linear optimization problem. We

[1] State constraints come into effect when it is desirable to avoid certain states. From a process engineering standpoint, state constraints can imply safety, technical or operational constraints. For instance, when operating a certain chemical reactor, it is important to ensure that the inlet and outlet pressures of the reactor stay within the prescribed safe operational range. State constraint is critical for Model Predictive Control (MPC) like controllers. However, RL in its original form is not naturally designed to handle such constraints as the goal of the agent is to score maximum reward and win. Certain constraint handling modifications are required to achieve this.

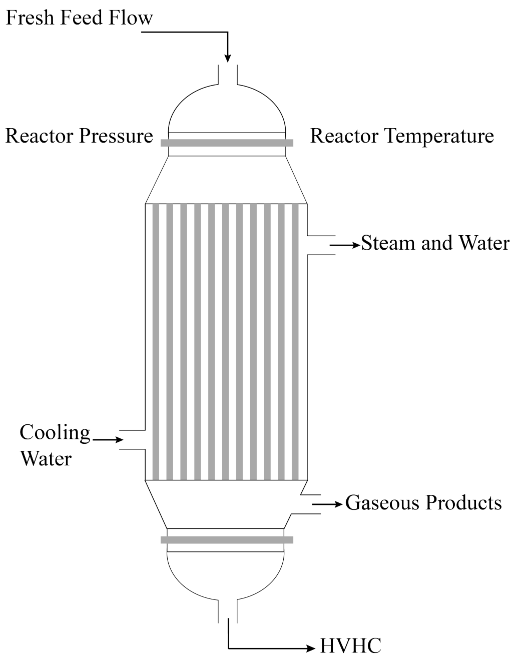


Fig. 1: Diagrammatic representation of a single reactor. The full unit is a two-staged system consisting of identical r_1 reactors in the first stage and r_2 reactors in the second stage. The reactors produce HVHC, a high value hydrocarbon which is also the key product of interest. HVHC production is determined by several factors such as catalyst activity factors, fresh feed flow values, reactor inlet pressures and reactor temperatures

use a suitable actor-critic RL architecture and compare our results with mathematical optimization solver based benchmark developed in MATLAB. Furthermore, we briefly evaluate the capabilities and features of Microsoft Project Bonsai (Microsoft Bonsai, 2020), an AI platform for designing autonomous systems, by implementing the solving methodology both in in-house code and on Bonsai platform. Major contributions of this work include the following:

- We demonstrate an application of RL for RTO problems in chemical processes. Our approach is distinct from traditional RTO methodologies and we propose potential benefits such as self-adaptiveness and fast inference time. Additionally, our implementation is generic and has the potential to be extended for analogous operations.
- We discover that RL can find good quality feasible solutions to the RTO problem. We observe that these solutions are typically comparable with the MATLAB benchmark. Furthermore, we put forward two primary advantages of RL over the optimization-based benchmark: adaptiveness and fast inference time.
- Previous research has noted that implementing RL in highly constrained environments is challenging (Pan et al., 2021), (Dalal et al., 2018). We confirm this and present one method of constraint handling - augmenting reward function with penalty functions during policy network training to penalize the RL agent and prevent the network from prescribing inputs or outputs that violate variable bounds or process constraints.
- We propose a training scheme to design adaptive RL agents that can generalize to unseen input scenarios and compare performance against the benchmark solutions.

The paper is structured as follows. In Section 2, we present the problem statement and elaborate upon decision variables, objective function and process constraints. We cover the details of adopted methodologies for problem solving in Section 3. Finally, in Section 4 and Section 5, we present our results and conclusions respectively.

2 Real-time Optimization Problem

In this section, we outline the constrained optimization problem. Please note that due to certain confidentiality reasons, we present a *restricted* version of the problem, providing only the necessary details and normalized data.

We aim to optimize a two-stage chemical reactor system for the production of a high-value hydrocarbon (*HVHC*). The system consists of r_1 reactors in the first stage and r_2 reactors in the second stage, designed to convert raw input gaseous fresh feed to HVHC in the presence of a catalyst and right physical conditions. A diagrammatic representation of a single reactor is shown in Figure 1. By adjusting controllable variables, such as reactor temperature, pressure, and the flow of fresh feed to individual reactors etc., it is possible to optimize the total HVHC yield of the unit. Hence, the objective is to identify the optimal values of these controllable variables such that the HVHC production is maximized, while adhering to variable bounds and process constraints. The reactor system is simulated through a thermodynamic first principles-based process model implemented in MATLAB. The model accounts for internal process constraints, such as mass balance, energy balance etc.. This high-fidelity, steady-state model takes manipulated variables and disturbance variables (catalyst activity factors) as inputs. The model produces the corresponding HVHC production, along with output variables such as the outlet composition of gaseous products and other performance indicators.

We provide further details about the manipulated variables, disturbance variables, output variables and process constraints below.

Manipulated Variables. The optimization of HVHC yield involves manipulation of 52 variables that are controllable, also known as manipulated variables (MV's). The reactor temperature, pressure, and inlet flow to the reactors are among the variables that can be tuned. Every MV is associated with a distinct lower and upper bound. We denote the MV's as a 52-dimensional vector and use the symbol m to represent it.:

$$m = \begin{bmatrix} m^1 \\ m^2 \\ \vdots \\ m^{52} \end{bmatrix} \tag{1}$$

, where $m^i\,(i = 1, 2...., 52)$ is the i^{th} element of m with $m^i \in [m^i_{min}, m^i_{max}]$. m^i_{min} and m^i_{max} are the lower and upper bounds, respectively, for each m_i, with $m^i_{max} > m^i_{min} \geq 0$.

Disturbance Variables. Disturbance variables (DV's) correspond to the activity factors of the catalyst in each reactor, and we represent them using a vector d of length 12, where each element d_i takes on a value in the range $[0, 1]$. We note that the catalyst activity factor may have a significant impact on the performance of the reactor system, and therefore, it is essential to consider them in our RL algorithm.

Output Variables. Output variables (OV's) are used to characterize the state of the system after chemical reactions in the reactors are completed. The OV's consist of measurable quantities such as compound-wise outlet composition, outlet temperature and pressure, key ratios in outlet gaseous products, safety indicators etc., and are determined for each individual reactor. A total of 228 variables constitute our OV space, each having corresponding lower and upper bounds. The OVs are represented by a vector o that has a length of 228. Each element o^i is assigned a value in the range $[o^i_{min}, o^i_{max}]$, where o^i_{min} and o^i_{max} are the lower and upper bounds for o^i respectively, with $o^i_{max} > o^i_{min} \geq 0$.

Process Constraints. Two process constraints are taken into account here. The first constraint, denoted as C_1, involves an upper bound on the total amount of raw gaseous fresh feed that can be supplied to the unit. The second constraint, referred to as C_2, requires that the inlet pressure of the unit should be greater than its outlet pressure.

We evaluate the performance of the RL-based RTO methodology for two graduated scenarios, namely *fixed-DV* and *dynamic-DV*. In the *fixed-DV* scenario, a fixed decision variables vector, d, is used to train the RL agent, which is then evaluated on the same once training is complete. Conversely, in the dynamic-DV scenario, random and admissible sets of DV's are employed at each iteration to train the agent. After training, the agent is assessed on unseen DV's to compare its performance with the MATLAB benchmark. By utilizing random DV's in the dynamic-DV approach, the RL agent can discover optimal MV's for each corresponding DV, learn underlying patterns, and generalize to unseen DV's. The goal is to create an RL-based optimization engine that can effectively optimize the system for any given d.

3 Methodology

In this section, we propose RL-based methodology to solve the problem introduced in the previous section. We first provide a brief overview of the fundamentals of RL. Subsequently, we describe the creation of the learning environment for the RTO problem. We then discuss in-house and Bonsai methodologies for solving the problem.

3.1 Reinforcement Learning

Figure 2 depicts the fundamental learning mechanism in RL. At time step t, the agent takes an action a_t and the environment transitions from state s_t to state s_{t+1}, resulting in an immediate reward r_{t+1}. The agent then uses the state information s_{t+1} and the immediate reward r_{t+1} to select the next action a_{t+1}, and the cycle continues.

The optimal policy, denoted by $\pi(a_t = a|s_t = s)$, is a mapping from states to actions that maximizes the value function $v_\pi(s)$, which represents the long-term expected sum of future rewards:

$$v_\pi(s) = \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty}\gamma^t r_{t+1}|s_t = s\Big], \tag{2}$$

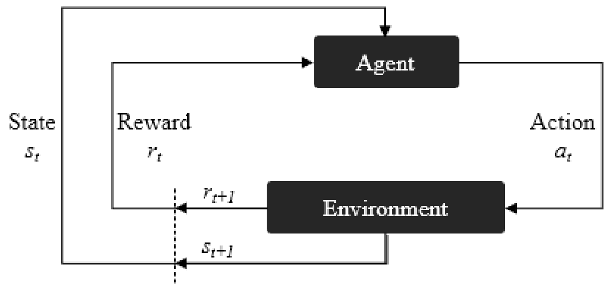


Fig. 2: Schematic of agent-environment interaction in RL: the agent at time step t, interacts with the environment, represented by the state s_t. Based on its policy $\pi(a_t|s_t)$, the agent takes an action a_t, receives a reward r_t and transitions to a new state s_{t+1} . The goal of the agent is to maximize the expected cumulative reward in the long run

where $\mathbb{E}[.]$ denotes the expected value of a random variable given that the agent follows policy π, and t is the time step. Here, the discount factor $\gamma \in [0, 1]$ determines the importance of future rewards, with $\gamma = 0$ corresponding to a greedy agent that only cares about the reward at the next time step, and $\gamma = 1$ corresponding to an agent that values all future rewards equally. It is important to note that each value function $v_\pi(s)$ is tied to a specific policy π, which dictates the agent's future trajectory through the space of states.

RL algorithms can be thought of as iterative updates to a policy that improve the associated value function for all states. If the agent properly adjusts the policy at each iteration, the policy will continue to improve, resulting in larger and larger values for the value function for any given state. It is natural to wonder whether the value function ever reaches its optimal value, $v_*(s)$. Bellman's optimality equation, a necessary condition in optimal control theory, provides an answer to this question:

$$v_*(s) = max \sum_{s',r} p\left(s', r|s, a\right)\left[r + \gamma v_*(s')\right] \tag{3}$$

Here, the transition probability $p\left(s', r|s, a\right)$ calculates the probability that the environment will transition to state s' with reward r given the current state s and action a. Bellman's optimality equation gives a set of nonlinear equations that, in theory, can be solved directly to produce the optimal value function $v_*(s)$ for a discrete set of actions and states.

In practice, however, these equations are often not directly solvable due to either a lack of knowledge about the environment's transitions or the state space being too large to allow for a reasonable solution. All RL algorithms are therefore approximate solutions to Bellman's optimality equation, and they address these limitations in different ways.

For continuous states and actions, the state and action spaces can be enormous, making it impossible to derive exact solutions for the optimal value function. In these cases, function approximation methods, such as neural networks, are often used. Specifically, we can parameterize the approximate value function with parameters θ as follows:

$$v(s) \cong v_\theta(s) \tag{4}$$

The goal of the RL method in this case is to estimate θ that produce a value function that is as close to the actual value function as possible, for example, in terms of reducing the mean squared error (MSE):

$$L(\theta) = \frac{1}{2} \sum_s \left[v_\theta(s) - v_*(s)\right]^2 \tag{5}$$

RL methods can be divided into two main categories: value-based methods and policy-based methods (Sutton et al., 2018). Value-based methods estimate the value function $v_\pi(s)$ or $v_*(s)$ directly, while policy-based methods estimate the policy $\pi(a|s)$ directly.

One popular value-based method is Q-learning, which estimates the action-value function $Q(s, a)$, which represents the expected return when starting in state s, taking action a, and then following the optimal policy thereafter. The Q-function can be expressed as:

$$Q_*(s, a) = E_{s'\sim p}\left[R_{t+1} + \gamma \max_{a'} Q_*(s', a')\right] \tag{6}$$

where $s' \sim p$ denotes the next state s' being drawn from the transition probability distribution p. Q-learning aims to find the optimal action-value function $Q_*(s, a)$ by iteratively updating an estimate $Q(s, a)$ according to the following update rule:

$$Q(s, a) \leftarrow Q(s, a) + \alpha\left[R_{t+1} + \gamma \max_{a'} Q(s', a') - Q(s, a)\right] \tag{7}$$

Here, α is the learning rate. The Q-learning algorithm converges to the optimal action-value function $Q_*(s, a)$ as $\alpha \longrightarrow 0$.

One popular policy-based method is policy gradient, which directly optimizes the policy by adjusting the parameters of the policy function. Policy gradient methods update the policy function by following the gradient of the expected return with respect to the policy parameters. The goal is to find the optimal policy $\pi_*(a|s)$ that maximizes the expected return:

$$\pi_*(a|s) = argmax_\pi E_{s'\sim p,r}\left[R_{t+1} + \gamma \sum_{a'} \pi(a'|s')Q_*(s', a')\right] \tag{8}$$

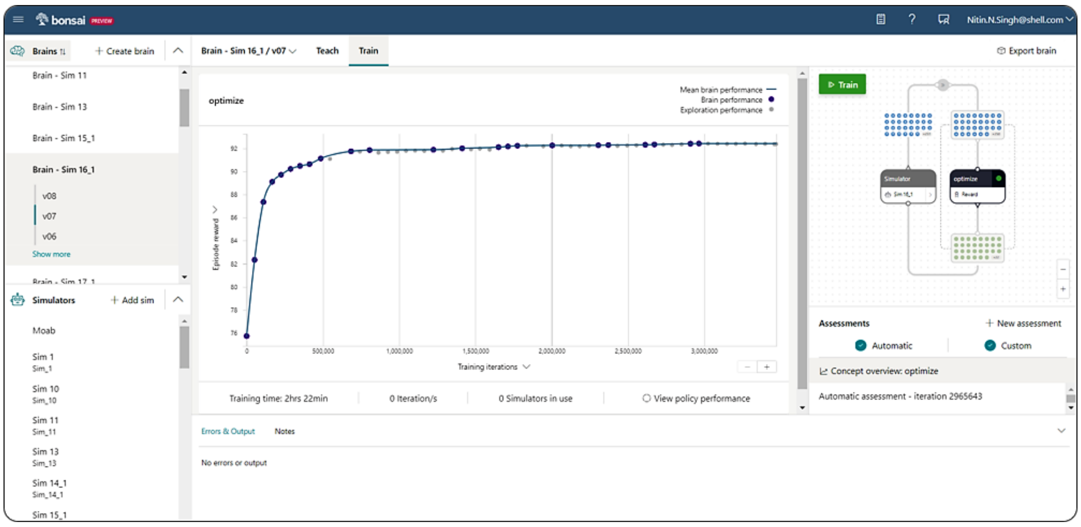


Fig. 3: Snapshot of Bonsai platform's home screen for demonstration purpose. Brain 'v01' is connected to the simulator 'Cartpole'. The simulator includes the step and reset functions. In the central panel, the graph shows the brain's gradual learning. The 'Teach' tab at the top navigates user to the Inkling file and enables its custom configuration

3.2 Learning Environment and Methodology Implementation

We use the first-principles based simulation model to construct the learning environment for the RL agent. A successful run of the model requires valid inputs: manipulated variables (m) and disturbance variables (d), and produces output variables (o), along with the HVHC production (Section 2). We note the correspondence between some of the terminologies used in our model and those used in RL. m corresponds to actions in RL terminology, while a combination of o and d correspond to state space. The total HVHC production forms one part of the reward for the agent[2].

The environment is designed based on the suggested methodology in (Slater, 2019). This setup consists of two key functions: `step` and `reset`. The `step` function takes the current state, s, and the action, a, as parameters and outputs the reward, r, and the subsequent state, s'. This is known as a *single transition step*, represented by the tuple $\{s, a, r, s'\}$. The `reset` function resets the agent to a default state when a terminal state or condition is reached, or in RL terminology, when a single episode finishes. Below, we elaborate upon our approaches of in-house implementation and Bonsai platform.

In-house Algorithm Implementation. An actor-critic DRL algorithm called DDPG (Lillicrap et al., 2015) is implemented in-house. This algorithm is suitable for handling continuous action and state spaces. The performance of DDPG depends critically on algorithm hyperparameters such as actor and critic learning rates, batch size, soft-update rate (τ), discount rate (γ), etc. No single set of hyperparameters can be assumed optimal for all problem types. To tune the performance of DDPG, we conduct hyperparameter optimization by setting up trial runs of the algorithm with different hyperparameter values and selecting the one that gives the best performance. We elaborate on our approach in Section 4.1.

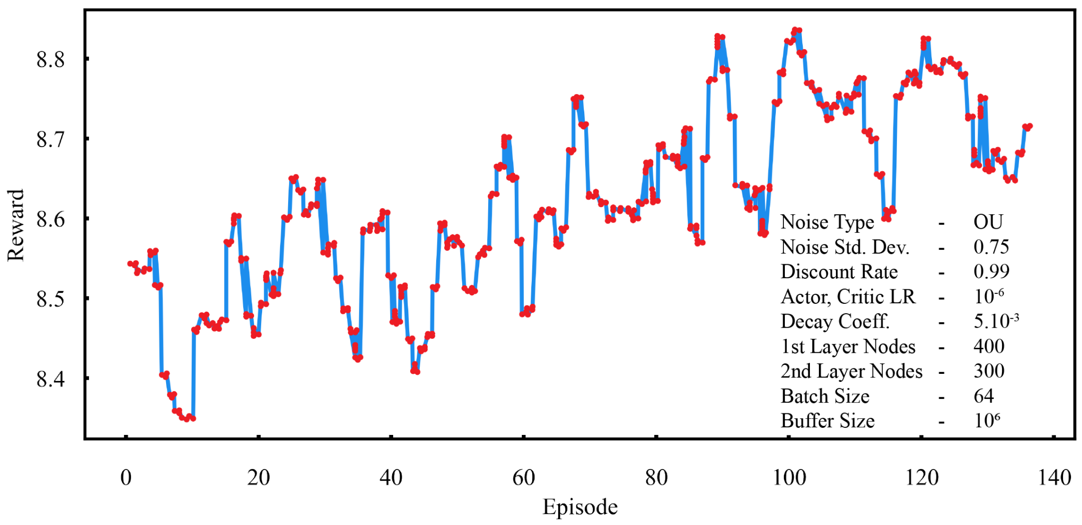
Bonsai Platform. The Bonsai platform by Microsoft is a low-code programming platform for creating autonomous systems using RL. It supports simulators built using MATLAB, Transys, AnyLogic, and provides techniques and infrastructure for model deployment and training. The platform's programming language, *Inkling*, is used to encode concepts into a model, which is then linked with a simulation of a real-world system and fed into the Bonsai AI Engine for training. The engine lays out the neural networks and tunes the hyperparameters for optimal training results. A significant advantage of the platform is its computational resources, allowing for many simulations to be run in parallel to reduce training time.

Bonsai's dashboard (Figure 3) enables users to examine all training agents, referred to as *Brains*, as well as their training state, online/offline assessment, and options for debugging, inspecting, and enhancing models. To get started, Microsoft provides support through starter Inkling files and online syntax documentation. The loading of the simulator, training, and assessment are all handled through button clicks on the Bonsai online platform. We integrated MATLAB simulator with Bonsai's *glue codes* to make it platform-compatible.

4 Results and Discussion

In this section, we present the key results. We first discuss the results from in-house implementation, followed by implementation in Bonsai platform.

[2] The other part of the reward signal arrives from a penalization scheme we employ for constraint handling, where a penalty value is subtracted from the primary reward value. Further elaborated in Section 4.1, 4.2



Noise Type	-	OU
Noise Std. Dev.	-	0.75
Discount Rate	-	0.99
Actor, Critic LR	-	10^{-6}
Decay Coeff.	-	5.10^{-3}
1st Layer Nodes	-	400
2nd Layer Nodes	-	300
Batch Size	-	64
Buffer Size	-	10^{6}

Fig. 4: Sample reward curve for DDPG in-house implementation. Best performing hyperparameter values are mentioned inside the figure

4.1 In-house Implementation

We implement an off-the-shelf actor-critic algorithm, Deep Determinstic Policy Gradient (DDPG) (Lillicrap et al., 2015) for solving our problem. DDPG is specifically suited for environments with continuous actions and observations space. In the in-house implementation, we only solve the fixed-DV problem, where we train and evaluate the DRL agent on the same disturbance variable vector d.

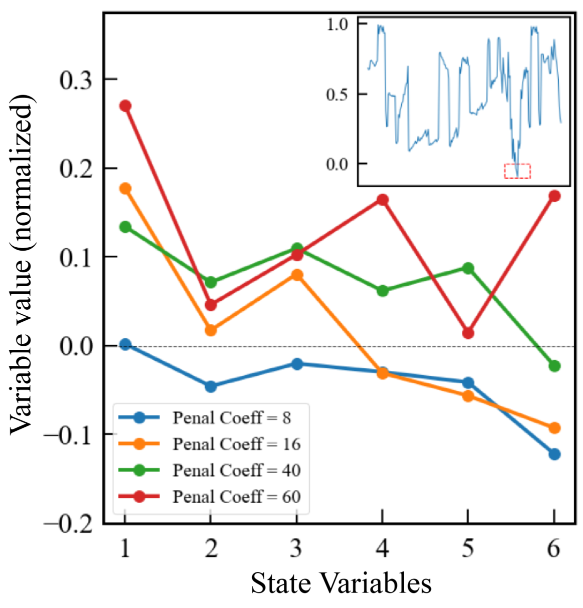
A common challenge that arises in employing RL for constrained environments is that of *invalid actions* - during training, an action is taken which might result in the value of a state variable to lie outside of its prescribed bounds (Nian et al., 2020). However, ensuring avoidance of constraint violations is critical in our case, as it can lead to risky unit operations. We implement a simple penalization scheme to deal with this issue. At each interaction with the environment, we assess the total sum of absolute constraint violations on the action and observation bounds, as well as on constraints C_1 and C_2 (Section 2). These violations are multiplied by constant scaling factors to balance the scales and subsequently subtracted from the observed reward (HVHC production) to arrive at the final reward value. Although we have used a particular choice of scaling factors, other options are equally viable. This scheme offers a possible way to incentivize the agent to refine its policy while simultaneously satisfying the constraints.

The performance of DDPG algorithm is highly reliant on the hyperparameter settings (Henderson et al., 2018). To optimize the hyperparameters, we conduct multiple runs of our implementation on a high-performance computer cluster using different hyperparameter configurations. The most suitable hyperparameter values and a sample training curve of the agent is shown in Figure 4. We observe that a low learning rate (10^{-6}) for both the actor and critic networks facilitates optimal learning of the agent. Although the penalization scheme implemented to avoid constraint violations is effective to some extent, it is not entirely foolproof, as some constraint violations may still yield high rewards. To address this issue, we have implemented a straightforward filtering approach to identify feasible solutions that meet the constraint requirements, i.e., solutions with a high reward and zero constraint violation. The best feasible solution identified has a reward value of 8.84 tonnes/day, while the MATLAB optimizer provides an optimal solution at 9.35 tonnes/day. It is essential to note that although our in-house solution may not be optimal, it provides a viable alternative that adheres to the constraints of our scenario.

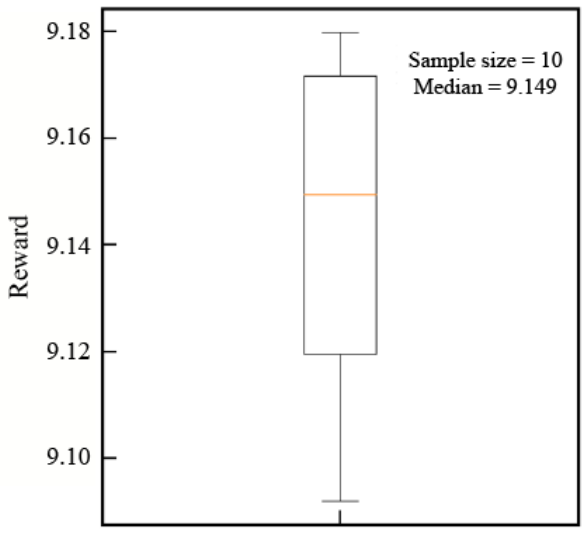
4.2 Bonsai Implementation

In this section, we discuss the implementation of our approach on Bonsai platform and present findings for fixed-DV and dynamic-DV case. We first discuss some key implementation details.

- **DRL Algorithm**. We use Proximal Policy Optimization (PPO) (Schulman et al., 2017) supported by the Bonsai platform. Similar to DDPG, PPO is also capable of handling continuous state and action spaces.
- **Action and Observation Scaling.** We utilize the upper and lower limits for state and action variables to normalize their values. Specifically, we normalize state space between [0, 1] and action space between [-1, 1] for assistance in policy training.
- **Delta Actions.** We mentioned in Section 1 about the static nature of real-time optimization problem. To induce pseudo-dynamics, we have experimented with an approach called *delta actions*, where we make small adjustments, referred to as *deltas*, to a default initial action in each iteration. We concatenate the resulting action array to the state array to provide the agent with a better understanding of the positive and negative changes in rewards based on the small changes in actions. This approach aims to improve the agent's ability to perceive the effects of its actions on the environment and thereby enhance its learning.



(a)



(b)

Fig. 5: (a) Impact of penalty coefficient on violation of state bounds. The plot shows how increasing the penalty coefficient limits the values of certain crucial and hard to contain state variables within the range of 0 to 1. (b) Distribution of final reward attained. This figure displays the median value of the reward achieved in 10 separate runs of the experiment on Bonsai platform

- **Constraint Handling.** We use a simple action clipping procedure to clip the action values such that they never violate the prescribed limits. We handle the constraints C_1, C_2 and limits on the states via a quadratic penalization method, where we first calculate the square of the total absolute constraint and bound violation. We multiply the resulting number with a 'temperature' factor that is equal to the iteration number during training. We do this in order to ensure that the agent is penalized heavily for violating constraints in the later iterations. Additionally, we control the total amount of penalty by multiplying it with a 'penalty coefficient', whose value is preselected. Penalty coefficient helps in controlling the scale of the total constraint violation.

4.2.1 Fixed-DV Scenario

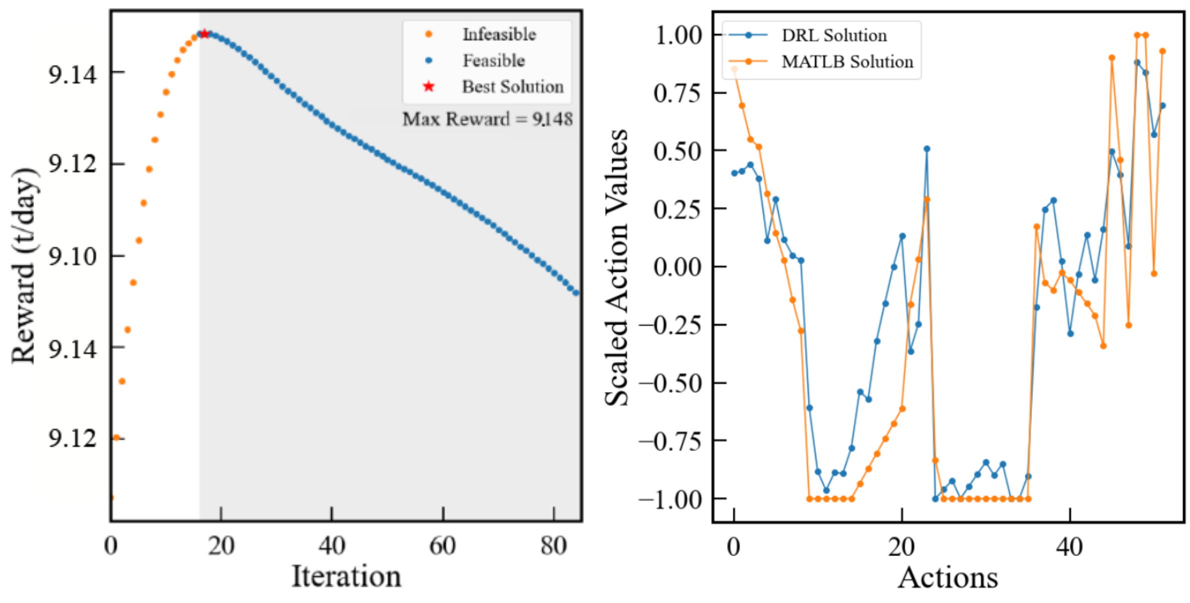
As mentioned above, we handle state variable bounds and constraints C_1, C_2, through a combination of quadratic penalization, penalty coefficient and temperature factor. The final penalty function is arrived upon through experimentation and observing on what performs best. We demonstrate it through an example. Figure 5a shows the effect of penalty coefficient on constraining certain critical and difficult to constrain state variables within their safe operational bounds i.e. between [0, 1] on the normalized scale (red box in the inset). We observe that an increase in the value of penalty factor leads to better chances of meeting variable bounds. Figure 5b summarizes the performance of Bonsai RL agent for the fixed-DV case. Based on 10 sample runs, we achieve a median reward value of 9.15 tonnes/day, with a maximum value at ~9.18. Compared to the in-house implementation, Bonsai platform clearly displays improved performance.

In Figure 6a, we display the assessment of a trained RL agent. We notice that at the beginning of the episode, the agent is within the infeasible domain and has a low reward value, however the agent gradually improves as the episode progresses. Finally, the agent is able to satisfy all the constraints (including bounds) and achieve a higher score. The best solution is labelled with a red marker and corresponds to the reward value of ~9.15 tonnes/day. Figure 6(b) compares the final optimal solution of Bonsai RL approach with the MATLAB benchmark. Interestingly, we observe close qualitative similarities between the suggested optimal solution by the two methodologies.

4.2.2 Dynamic-DV Scenario

In this section, we report the results for the dynamic-DV scenario, where we train the agent using a sets of training disturbance variable vector d and evaluate its performance on a distinct unseen disturbance variable vector. We construct 70 valid vectors d, from which the agent randomly selects one for each episode during training. We reserve a separate set of 30 d vectorsfor evaluation. The rest of the implementation details are similar to those of the fixed-DV scenario as listed in Section 4.2.

Figure 7 presents a comparison of our results with the MATLAB benchmark. The main panel of Figure 7(a) depicts the frequency distribution of solutions on test sets of d from the Bonsai framework in blue and those from the MATLAB benchmark in red. Our methodology achieves a median of 9.039 tonnes/day, while the MATLAB benchmark performs better with a median



(a) **(b)**

Fig. 6: (a) Performance of a trained Bonsai RL agent. The agent starts in an infeasible domain. Despite a low initial reward, the agent gradually improves and eventually satisfies all constraints, including bounds, achieving a maximum reward of ~9.15 tonnes/day. (b) Comparison of Bonsai RL and MATLAB benchmark solutions. The results show a high degree of qualitative similarity, suggesting the potential efficacy of RL for solving complex optimization problems

of 9.33 tonnes/day. We observed in this case that final Bonsai solutions consistently allowed for some improvement - the raw fresh feed consumption by the unit, as given by C_1 is not maximally utilized. We therefore could, manually *refine* the solutions. We equally distribute the 'leftover' amount of raw fresh feed among all the reactors, and observe that the resulting solutions achieve a higher HVHC production. Figure 7(b) shows the frequency distribution of improved solutions, resulting in an increase in median reward to 9.179 tonnes/day.

5 Conclusion

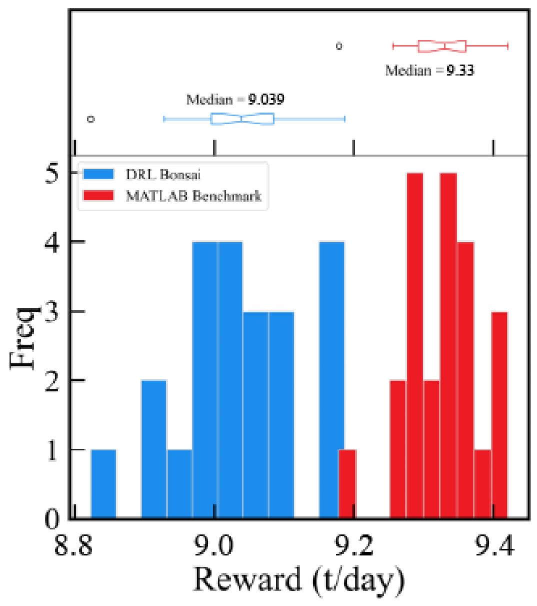
In conclusion, this work presents a proof of concept study exploring the application of RL for RTO in a catalytic reactor system. The challenges of dealing with high-dimensional action and state spaces and satisfying state constraints in industrial processes are discussed. The study aims to identify the optimal values of controllable variables to maximize the production of a high-value hydrocarbon while adhering to variable bounds and process constraints. The proposed RL-based approach is compared with a mathematical optimization-based benchmark developed in MATLAB, and the results show that RL can find good quality feasible solutions to the RTO problem with potential advantages of adaptiveness and fast inference time. The study also presents one method of constraint handling through augmenting reward function with penalty functions during policy network training to prevent the network from prescribing inputs or outputs that violate variable bounds or process constraints. The work contributes to the exploration of the application of RL in the process engineering and control domain, and the proposed approach has the potential to be extended for analogous operations. Furthermore, the study briefly evaluates the capabilities and features of Bonsai, an AI platform for designing autonomous systems, by implementing the solving methodology both in-house code and on the Bonsai platform.

Acknowledgments This work has been funded through the Shell.ai Sciences Innovation Programme. The authors would like to thank Hossein Khadivi Heris (Senior Applied AI Engineer, Microsoft) for his help with the Microsoft Bons.ai platform and productive discussions.

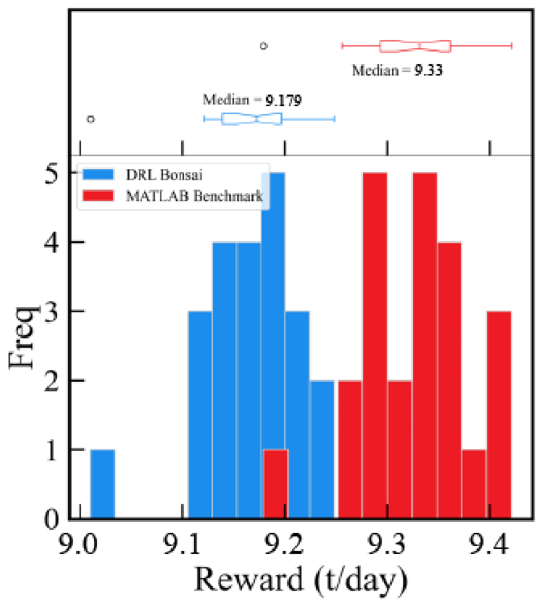
Funding Statement The authors would like to acknowledge the funding of the Shell.ai Futures Programme which has enabled funding for this research.

Competing Interests None

Data Availability Statement Due to the confidential nature of the data used in this project, access to the data and source code cannot be provided.



(a)



(b)

Fig. 7: Comparison of Bonsai framework and MATLAB benchmark for dynamic-DV scenario. (a) Frequency distribution of solutions in blue from the Bonsai framework and in red from the MATLAB benchmark. Our methodology has a median of 9.039 tonnes/day, while the MATLAB benchmark performs better with a median of 9.33 tonnes/day. (b) Frequency distribution of improved solutions resulting from manual refinement. The median reward value increases to 9.179 tonnes/day

References

Silver, **D** , **Huang**, **A**, **Maddison**, **C.J**, **Guez**, **A**, **Sifre**, **L**, **Driessche**, **G.V.D**, **Schrittwieser**, **J**, **Antonoglou**, **I**, **Panneershelvam**, **V**, **Lanctot**, **M**, **Dieleman**, **S**, **Grewe**, **D**, **Nham**, **J**, **Kalchbrenner**, **N**, **Sutskever**, **I**, **Lillicrap**, **T.P**, **Leach**, **M**, **Kavukcuoglu**, **K**, **Graepel**, **T**, **Hassabis**, **D** (2016) Mastering the game of Go with deep neural networks and tree search. *Nature 529*, 484–489.

Silver, **D**, **Schrittwieser**, **J**, **Simonyan**, **K**, **Antonoglou**, **I**, **Huang**, **A**, **Guez**, **A**, **Hubert**, **T**, **Baker**, **L**, **Lai**, **M**, **Bolton**, **A** (2017) Mastering the game of go without human knowledge. *Nature 570*, 354–359.

Mnih, **V**, **Kavukcuoglu**, **K**, **Silver**, **D**, **Graves**, **A**, **Antonoglou**, **I**, **Wierstra**, **D**, **Riedmiller**, **M** (2013) Playing Atari with Deep Reinforcement Learning. *arXiv*, 1312.5602.

Mnih, **V**, **Kavukcuoglu**, **K**, **Silver**, **D**, **Graves**, **A**, **Antonoglou**, **I**, **Wierstra**, **D**, **Riedmiller**, **M** (2017) Deep reinforcement learning framework for autonomous driving. *Electronic Imaging 19*, 70–76.

Xiong, **Z**, **Liu**, **X**, **Zhong**, **S**, **Yang**, **H**, **Walid**, **A** (2018) Practical deep reinforcement learning approach for stock trading. *arXiv*, 1811.07522.

Spielberg, **S**, **Tulsyan**, **A**, **Lawrence**, **N.P**, **Loewen**, **P.D**, **Bhushan**, **R.G**. (2019) Toward self-driving processes: A deep reinforcement learning approach to control. *AIChE Journal 65*, 1547–5905.

Ernst, **D** , **Glavic**, **M**, **Capitanescu**, **F**, and **Wehenkel**, **L** (2008) Reinforcement learning versus model predictive control: A comparison on a power system problem. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 39(2)*, 517–529.

Nian, **R**, **Liu**, **J**, **Huang**, **B** (2020) A review on reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering, 139*, 106886.

Kaelbling, **L. P**, **Littman**, **M. L**, **Moore**, **A. W** (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research, 4*, 237-285.

Powell, **KM**, **Machalek**, **D** and **Quah**, **T** (2020) Real-time optimization using reinforcement learning. *Computers & Chemical Engineering, 143*, 107077.

Pan, **E**, **Petsagkourakis**, **P**, **Mowbray**, **M**, **Zhang**, **D** and **Rio-Chanona**, **EA** (2020) Constrained model-free reinforcement learning for process optimization. *Computers & Chemical Engineering, 154*, 107462.

Microsoft Bonsai, (2020) Project Bonsai. *https://docs.microsoft.com/en-us/bonsai/product/.*

Dalal, **G**, **Dvijotham**, **K**, **Vecerik**, **M**, **Hester**, **T**, **Paduraru**, **C** and **Tassa**, **Y** (2018) Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757.*

Sutton, **RS** and **Barto**, **AG** (2018) Reinforcement learning: An introduction. *MIT press.*

Slater, **N** (2019) How to create a custom environment for reinforcement learning. *https://ai.stackexchange.com/questions/12577/how-to-create-a-custom-environment-for-reinforcement-learning.*

Lillicrap, **TP**, **Hunt**, **JJ**, **Pritzel**, **A**, **Heess**, **N**, **Erez**, **T**, **Tassa**, **Y**, **Silver**, **D** and **Wierstra**, **D** (2015) Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971.*

Henderson, **P**, **Islam**, **R**, **Bachman**, **P**, **Pineau**, **J**, **Precup**, **D**, **Meger**, **D** (2018). Deep reinforcement learning that matters. *In Proceedings of the AAAI conference on artificial intelligence, 32*, 1.

Schulman, **J**, **Wolski**, **F**, **Dhariwal**, **P**, **Radford**, **A**, **Klimov**, **O** (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347.*