Digital Chemical Engineering 17 (2025) 100277

Contents lists available at ScienceDirect

Digital Chemical Engineering

journal homepage: www.elsevier.com/locate/dche

Original article

Utilizing reinforcement learning in feedback control of nonlinear processes with stability guarantees

Arthur Khodaverdian [a], Xiaodong Cui [a], Panagiotis D. Christofides [a,b,*]

[a] Department of Chemical and Biomolecular Engineering, University of California, Los Angeles, CA 90095, USA
[b] Department of Electrical and Computer Engineering, University of California, Los Angeles, CA 90095-1592, USA

ARTICLE INFO

Keywords:
Reinforcement learning
Twin delayed DDPG
Hamilton–Jacobi–Bellman equation
Model predictive control
Closed-loop stability guarantees
Process control

ABSTRACT

This work explores the implementation of reinforcement learning (RL)-based approaches to replace model predictive control (MPC) in cases where practical implementations of MPC are infeasible due to excessive computation times. Specifically, with the use of externally enforced stability guarantees, an RL-based controller that is trained to optimize the same cost function as the MPC with a long horizon that achieves the desirable closed-loop performance can serve as a potentially more appealing real-time option as opposed to using the same MPC with a shorter horizon. A benchmark nonlinear chemical process model is used to demonstrate the feasibility of this RL-based framework that simultaneously guarantees stability and enables improvements in computational efficiency and potential control quality of the closed-loop system. To explore the influence of the RL training method, two RL algorithms are explored, with one imitation learning method used as a reference.

1. Introduction

Model Predictive Control (MPC) is a broadly applicable framework that enables finely customizable control policies through the use of constrained numerical optimization. Due to its core operating principle involving the construction of trajectories using an internal process dynamics model, MPC can account for the interactions between process variables over a horizon, resulting in high-quality control (Qin and Badgwell, 2003). A necessary consequence of this is poor scalability, as explicitly calculating these interactions across the various scenarios considered during the optimization process is a computationally intensive process. Nonlinear constrained MPC problems are NP-hard, whereas linear convex cases have a polynomial-time complexity of roughly $\mathcal{O}\left(n^4\right)$ (Pardalos and Vavasis, 1991; Peng et al., 2024). Even for more efficient methods, such as Sequential Least Squares Programming (SLSQP), the subproblems alone require $\mathcal{O}\left(n^2\right)$ or $\mathcal{O}\left(n^3\right)$ time (Gill et al., 1979; Kraft, 1988). Thus, MPC is burdened with poor scalability due to its polynomial-time complexity scaling laws.

To tackle this issue, research has been focused on a mix of solutions that can be roughly categorized as either reduction of the problem's scale or as optimization of the underlying computational framework. Reducing the problem scale with minimal performance losses is a viable and generally useful method to apply if possible; however, this approach fails to tackle the underlying scalability problem and is best suited for cases where a system is only barely infeasible, such that the application will reduce the computation time enough to allow for real-time application (Tomasetto et al., 2025; Alora et al., 2023; Zarrouki et al., 2023). Optimization of the framework is an approach that directly improves these scaling laws. Thus far, some improvements have been found, but there remain issues in ease of applicability, generalizability, and quantifying the impact on these scaling laws (Meng et al., 2024; Yaren and Kizir, 2025; Adabag et al., 2024). The latter point is particularly important, as the system may perform better by some roughly constant multiple but can still trend towards a particular order-of-magnitude scaling law as the model increases in size, which would mean that the underlying scaling issue is not resolved.

Within this research exists a subcategory of approximate methods that use neural networks (NN) to approximate MPC behavior or its model to achieve their goals. Research has demonstrated the validity of using NNs as a replacement for the system dynamics used inside of MPC, as the NN is capable of solving the dynamics with significantly less time and resources while also utilizing its data-driven properties to match – and in some highly complex cases, surpass – first-principles models in predictive accuracy (Wu et al., 2019; Gordon et al., 2024; Patel et al., 2025; Alsmeier et al., 2024; Ren et al., 2022; Macmurray and Himmelblau, 1995). Unfortunately, as a consequence of being used inside MPC, the overall polynomial-time complexity issue still exists due to aforementioned polynomial-time subproblems and the iterative methodology.

* Corresponding author at: Department of Chemical and Biomolecular Engineering, University of California, Los Angeles, CA 90095, USA.
E-mail address: pdc@seas.ucla.edu (P.D. Christofides).

https://doi.org/10.1016/j.dche.2025.100277
Received 23 October 2025; Received in revised form 26 November 2025; Accepted 26 November 2025
Available online 3 December 2025

Thus, research has focused on using NNs as a direct replacement for MPC, bypassing the iterative process entirely. The research thus far has been favorable, demonstrating that this idea is feasible for various processes, but details on scalability and strict constraint satisfaction are lacking (Lucia and Karg, 2018; Bonzanini et al., 2020; Ahn et al., 2022; Gonzalez et al., 2024). In our prior works, both issues have been explored, demonstrating a framework with strict stability guarantees that worked even for a large-scale, highly nonlinear process (Khodaverdian et al., 2025b, 2026). These works highlighted a core issue with NN-based control—data: as systems scale, the data required to adequately represent the operating region also scales.

Motivated by this problem, this work explores the use of Reinforcement Learning (RL) as a lightweight training alternative that does not require the use of MPC in the data generation process. As opposed to behavioral cloning methods, such as using gradient descent on the mean squared error loss between the NN's control action and the MPC's control action (as was done previously), RL methods can either utilize the first-principles model directly or interact with a physical system to explore the state-space and learn. The resulting trained controller is applied to a benchmark chemical process in order to demonstrate the feasibility of the controller within this stability-guaranteeing framework. Additionally, metrics during and after the training process are explored to observe the practical benefits of using RL as opposed to supervised learning. This analysis is explored with two different RL algorithms in order to assess the impact of the RL algorithm itself on the framework's capabilities and practicality.

2. Preliminaries

2.1. Notation

The transpose of a vector x, set of real numbers, set difference, functions, and piecewise-constant functions with period Δ are denoted by $x^\top$, $\mathbb{R}$, $\Omega_1 \backslash \Omega_2$, $f(\cdot)$, and $S(\Delta)$ respectively, where both f and S are arbitrary denotations. The initial instance of time (i.e., where $t = 0$) is denoted t_0, whereas arbitrary reference instances of time are denoted t_k.

2.2. Class of systems

This paper considers systems described by nonlinear first-order ordinary differential equations (ODEs) of the form:

$$\dot{x} = F(x,u) = f(x) + g(x)u = f(x) + \sum_{i=1}^{m} g_i(x)u_i \tag{1}$$

For these systems, the state vector $x = \left[x_1, x_2, \ldots, x_n\right]^\top \in \mathbb{R}^n$ is the vector representation of all relevant state variables for the process. These states are assumed to be measured at fixed sampling intervals of length Δ, as is standard for state feedback control. Similarly, the control input vector $u = \left[u_1, u_2, \ldots, u_m\right]^\top \in \mathbb{R}^m$ is the vector representation of all relevant variables that are applied as control actions. Unlike the system states, the control actions are bounded as a means to account for physical limits that real actuators would face. The lower ($u_{i,\min}$) and upper ($u_{i,\max}$) bounds on the control actions form the boundary of the set of valid control actions, defined as:

$$U := \left\{ u \in \mathbb{R}^m \middle| \begin{matrix} u = \left[u_1, u_2, \ldots, u_m\right]^\top \\ u_{i,\min} \le u_i \le u_{i,\max} \\ \forall\ i = 1,2,\ldots,m \end{matrix} \right\} \subset \mathbb{R}^m \tag{2}$$

Additionally, we utilize the deviation variable form of the system as a means to render the origin of the open-loop system (i.e., a system of the form shown in Eq. (1) where $u_i = 0\ \forall\ i = 1,2,\ldots,m$) as a steady state, thus ensuring $f(0) = 0$ without loss of generality. In other words, we treat $F(0,0) = 0$. We further assume that systems satisfying Eq. (1) have sufficiently smooth vector functions for $f(\cdot)$ and $g_i(\cdot)\ \forall\ i = 1,2,\ldots,m$.

2.3. Stabilizability assumption

The core assumptions that ensure stabilizability are collectively referred to as the stabilizability assumption. The stabilizability assumption consists of two main assumptions. The first is that we assume the existence of a sufficiently smooth explicit feedback control law that renders the origin of the system described by Eq. (1) exponentially stable. This controller is referred to as the reference stabilizing controller, or reference controller, as a shorthand.

$$\boldsymbol{\Phi} : \mathbb{R}^n \to U \tag{3}$$

$$u(x) = \boldsymbol{\Phi}(x) \tag{4}$$

The second is the assumption that there exists a sufficiently smooth Lyapunov function $V(x)$ which, when applied to the closed-loop system utilizing the reference controller for all x bounded by an open neighborhood near the origin, denoted D, satisfies the following inequalities:

$$c_1 |x|^2 \le V(x) \le c_2 |x|^2 \tag{5a}$$

$$\frac{\partial V(x)}{\partial x} F(x, \boldsymbol{\Phi}(x)) \le -c_3 |x|^2 \tag{5b}$$

$$\left| \frac{\partial V(x)}{\partial x} \right| \le c_4 |x| \tag{5c}$$

$$c_i > 0\ \forall\ i \in \{1,2,3,4\} \tag{5d}$$

The remaining parts of the stabilizability assumption are derived from the sufficiently smooth assumption for the system dynamics mentioned earlier. This implies Lipschitz continuity for $V(x)$, $\boldsymbol{\Phi}(x)$, and $F(x, \boldsymbol{\Phi}(x))$. Additionally, because $\boldsymbol{\Phi}(x)$ is bounded and x is implied to be bounded through the restriction of $x \in D$, we can say that $F(x, \boldsymbol{\Phi}(x))$ is bounded. Finally, note that the product of two continuously differentiable functions yields a function that is at least continuously differentiable. Thus, the stabilizability assumption implies the existence of positive constants M_F, L_x, L'_x that ensure, for all $x, x' \in D$ and $u \in U$, that the following inequalities are satisfied:

$$\left| F(x', u) - F(x, u) \right| \le L_x |x - x'| \tag{6a}$$

$$|F(x,u)| \le M_F \tag{6b}$$

$$\left| \frac{\partial V(x)}{\partial x} F(x,u) - \frac{\partial V(x')}{\partial x} F(x',u) \right| \le L'_x |x - x'| \tag{6c}$$

2.4. Lyapunov-based model predictive control

The stabilizability assumption can be applied to MPC to yield the Lyapunov-based MPC (LMPC) that solves for optimal control while ensuring closed-loop stability within D (Mhaskar et al., 2006).

$$\mathcal{J} = \min_{u \in S(\Delta)} \int_{t_k}^{t_k + N\Delta} L(\tilde{x}(t), u(t))\ \mathrm{d}t \tag{7a}$$

$$\text{s.t.} \quad \dot{\tilde{x}}(t) = F(\tilde{x}(t), u(t)) \tag{7b}$$

$$u \in U,\ \forall\ t \in [t_k, t_k + N\Delta) \tag{7c}$$

$$\tilde{x}(t_k) = x(t_k) \tag{7d}$$

$$\dot{V}(\tilde{x}(t_k), u(t_k)) \le \dot{V}(\tilde{x}(t_k), \boldsymbol{\Phi}(\tilde{x}(t_k))) \tag{7e}$$

In this formulation, the optimization takes place over a horizon of length $N\Delta$, with Δ denoting the controller's sampling period and N the horizon's sampling step count. This formulation uses sample-and-hold control, as continuous-time control is infeasible for real-world processes. For simplicity, the sampling interval for both the controllers and state measurements is treated as equivalent. Eq. (7a) denotes an arbitrary cost as a function of the control actions and estimated states over the horizon. Eq. (7b) represents the process dynamics, which are used

for numerical integration during optimization to predict how the states of the closed-loop system evolve over the horizon. Eq. (7d) enforces an initialization step where the sensor readings are used as a ground truth initial state, and Eq. (7c) enforces the control bounds. Eq. (7e) denotes the stability constraint; the implementation of the stabilizability assumption. This constraint ensures that the system is at least as stabilizing as the reference controller. An alternative form is provided:

$$\dot{V}\left(\tilde{x}\left(t_k\right), u\left(t_k\right)\right) \leq -\alpha V\left(\tilde{x}\left(t_k\right)\right) \tag{8}$$

This formulation uses the properties of the Lyapunov function from Section 2.3 as opposed to using the reference controller. Using a positive constant α to control the strength of this constraint, this formulation allows for stability guarantees without needing to directly apply the reference controller.

Remark 1. Referring to Eqs. (7e) and (8), we note that they can be thought of as constraints imposed on the MPC at the sampling time t_k to ensure that the closed-loop state converges towards the origin and further guarantee that the stability region of the Lyapunov-based controller (expressed in terms of a level set of $\dot{V}$ embedded in D) becomes a stability region of the LMPC. These constraints originate from properties of Lyapunov functions that satisfy the stabilizability assumption. Notably, Eq. (7e) is derived from the negative upper bound of $\dot{V}$ for the process under stabilizing control as shown in Eq. (5b), and Eq. (8) is derived from the fact that Lyapunov functions are positive outside of the origin, where they are 0, which is a general property of Lyapunov functions.

Remark 2. LMPC does not pose constraints on the form of the cost function. Although this paper will use a quadratic cost function, other formulations are supported. Economic MPC is one such modification that can enable enhanced cost-efficiency of processes in a manner that supports time-varying economics (Khodaverdian et al., 2025a).

Remark 3. Eq. (7e) is only applied at t_k because this formulation is a receding horizon LMPC, where only the first control input from the solution is applied. After applying the first solution for one sampling interval, the LMPC problem is re-solved. This approach relaxes the constraints of the optimization problem, allowing for faster solutions, but comes at the cost of marginally reduced accuracy of the cost-optimal trajectory.

Remark 4. Consider an LMPC formulation that satisfies the design above. We consider two cases of this LMPC: one with a long horizon and one with a short horizon. The long-horizon case is used purely for reference of what the truly optimal behavior would be (MPC optimal control action calculation improves with increased horizon length), as this case would take longer to calculate than the sampling interval, thereby making it infeasible for real-time control. The short-horizon case is a suboptimal solution relative to the long-horizon LMPC that is, however, faster to solve in real-time. This short-horizon LMPC can thus be used as a fallback controller as a means to enforce the stability guarantees for the closed-loop system at the expense of poor cost optimality.

2.5. Reinforcement learning

Reinforcement learning (RL) is a framework in which an agent interacts with an environment via a policy—a mapping from states to actions—to record states (s), actions (a), and rewards (r) to use to learn how to adapt its policy to maximize cumulative rewards. These values are typically stored in a replay buffer for use in training as $D := \left\{\left(s_i, a_i, r_i, s_i'\right)\right\}_{i=1}^{N}$. Here, s' denotes the future state that results from taking action a at state s. To guide learning, the agent often uses a value function that estimates long-term returns. There are two common forms of the value function, the state-value function denoted V and the action-value function denoted Q.

The state-value function is a function that quantifies how good the current state is. In the context of a continuous deterministic system, it yields the discounted cumulative reward starting at a provided state for some given policy:

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma V^{\pi}\left(s'\right) \tag{9}$$

where $0 < \gamma < 1$. This equation is commonly referred to as the Bellman Equation for the state-value function. The optimal state-value function, also known as the Bellman optimality equation, is found for the case where the action used maximizes the state-value function for all states.

$$V(s) = \max_{a}\left[r(s,a) + \gamma V\left(s'\right)\right] \tag{10}$$

Once this optimal state-value function is known, the optimal policy is known.

$$\pi(s) = \arg\max_{a}\left[r(s,a) + \gamma V\left(s'\right)\right] \tag{11}$$

Thus, there is a relationship between the state-value function and the policy that allows for the iterative solution of the optimal policy. In fact, there are two common iterative methods for doing this: value iteration and policy iteration.

In value iteration, the value function is first fully optimized by iteratively applying the Bellman optimality equation from Eq. (10) to the current form of the state-value function for all states. Once enough sweeps of the state-space are done such that the state-value function is now optimal, the optimal policy is extracted from it greedily via Eq. (11). In other words, the following equation is applied iteratively

$$V_{i+1}(s) = \max_{a}\left[r(s,a) + \gamma V_i\left(s'\right)\right] \tag{12}$$

where the subscript i denotes the iteration number for the value function. This method in particular is model-dependent due to the max operation, which requires an explicit way to determine the reward function and the solving of s'.

Similarly, policy iteration utilizes the Bellman equation from Eq. (9) for the current policy guess instead of the optimal equation from Eq. (10). This enables faster sweeps through the state-space, but requires that after the state-value function converges, the policy must be updated greedily via Eq. (11). Then, the entire iterative loop must be done again with this new policy, continuing a cycle of iteratively improving the state-value function and the policy until both converge. The benefit of both methods is that they guarantee monotonic convergence to the optimal value function, and thus optimal policy. The consequence of this is that both methods require the ability to do a full sweep of the state-space and action-space, neither of which is feasible for continuous-time systems, resulting in a loss of guarantees. Additionally, the need for a perfect process model complicates the applicability to systems whose real-world dynamics may not fit well with simple models.

An alternative would be to use model-free approaches, which enable learning the model and extracting optimal results through data and interaction with the system. To do this, the action-value function Q is commonly used as opposed to the state-value function V. This is done because the state-value function inherently assumes a fixed policy and will thus average the impact of various actions taken for a given state in the data. On the other hand, the action-value function explicitly uses state–action pairs, which avoids this issue entirely. Further, the training can be done in two forms, on-policy or off-policy.

On-policy training utilizes the current policy when updating the Q function, whereas off-policy training works with any source for a given action. An example of each would be SARSA and Q-learning, as shown below. The first equation represents the iterative value improvement

step used in SARSA, which is on-policy, and the second represents Q-learning, which is off-policy, and can be written as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q\left(s',a'\right) - Q(s,a)\right] \tag{13}$$

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q\left(s',a'\right) - Q(s,a)\right] \tag{14}$$

In both cases, the actions are implied to be from some greedily derived policy, with the exception of the max action used in Q-learning. These learning methods can be applied in a step-by-step manner to a system over as many episodes as needed until convergence. Q-learning's use of the max operator leads it towards being more sample efficient, but comes with the consequence of overestimation bias that can lead to risky actions. Further, both methods rely on a policy that is derived from the value function itself, which remains a computationally intensive process. This is mitigatable by using an Actor–Critic framework, where the policy, referred to as the Actor, is isolated from the value function and is independently iteratively improved.

2.6. Hamilton–Jacobi-bellman-based RL control

Hamilton–Jacobi–Bellman (HJB)-based reinforcement learning control formulates the control problem as solving the HJB optimality equation. In this framework, the value function is defined as follows:

$$V^*(x) = \min_{u(\cdot)}\left\{\int_t^\infty r(x(\tau),u(\tau))\,d\tau\right\} \tag{15}$$

The $V^*(x)$ quantifies the best long-term cost achievable starting from a given state with the following HJB condition:

$$\min_u H\left(x,u,V^*\right) = \min_u\left\{r(x,u) + \frac{\partial V^*}{\partial x}F(x,u)\right\} = 0 \tag{16}$$

Consequently, the optimal policy is the action that minimizes this Hamiltonian at each state:

$$\pi^*(x) = \arg\min_u H\left(x,u,V^*\right) \tag{17}$$

To ensure a fair comparison between LMPC and RL, the instant cost r in the RL formulation is chosen to match the quadratic cost function used in LMPC:

$$r(x,u) = L(x,u) = x^\top W_x x + u^\top W_u u \tag{18}$$

so that the HJB Hamiltonian becomes

$$H(x,u,V^*) = x^\top W_x x + u^\top W_u u + \frac{\partial V^*}{\partial x}\big(f(x) + g(x)u\big). \tag{19}$$

Accordingly, the HJB condition (Eq. (16)) can be solved under the stationarity conditions (Lewis et al., 2012):

$$\pi^*(x) = -\frac{1}{2}W_u^{-1}g(x)^\top\frac{\partial V^*(x)}{\partial x} \tag{20}$$

Since the exact optimal value function $V^*(x)$ is not available in closed form, we introduce a differentiable critic network $V_w(x)$ parameterized by neural network weights w. The critic is trained to minimize the mean squared residual of the HJB equation evaluated at sampled states $\left\{x\left(t_i\right)\right\}_{i=1}^N$. Specifically, the training objective and the weight update are given as follows:

$$\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^N\left(r\left(x\left(t_i\right),u\left(t_i\right)\right) + \left(\frac{\partial V_w}{\partial x}\right)\left(f\left(x\left(t_i\right)\right) + g\left(x\left(t_i\right)\right)u\left(t_i\right)\right)\right)^2 \tag{21a}$$

$$w \leftarrow w - \alpha_w \nabla_w \mathcal{L}(w) \tag{21b}$$

Here, $\mathcal{L}(w)$ denotes the loss function, and α_w is the learning rate that determines the step size of gradient descent.

Remark 5. The optimal control policy obtained from the HJB condition does not explicitly account for stability constraints. To enhance the accuracy of the approximated optimal value function, the neural network performance could be further improved by incorporating stability criteria in the training process; however, this aspect is not within the scope of the present study.

Remark 6. In the practical implementation, the instantaneous cost (or reward) r used in HJB-RL training is evaluated for a discrete time representation of the system. In this discrete-time representation, the LMPC's integral cost is approximated as a sum over a finite number of steps of size Δ where the state term in the cost function is replaced with the future state ($x(t_k + \Delta)$). Likewise, the reward function used in the HJB-RL training uses the future state ($x(t_k + \Delta)$) as opposed to the current state ($x(t_k)$).

2.7. Twin delayed DDPG-based RL control

While Actor–Critic methods are traditionally on-policy, there exist modern forms that maintain the core idea of having a separate Actor and Critic while enabling off-policy reinforcement learning. Twin Delayed DDPG, also known as TD3, is an off-policy Actor–Critic method that is designed for use with deterministic continuous systems. The algorithm is an evolution of the Deep Deterministic Policy Gradient (DDPG) algorithm with three modifications that aim to dampen extreme behaviors that are common in DDPG: the addition of clipped policy noise, the addition of a second Q function, and the addition of update delays for the policy and target networks. With these changes, the algorithm lessens the risk of overestimation of the Q function, which in theory should make the training more stable than DDPG. These algorithms represent both the Actor and Critic as neural networks, where the target networks refer to a set of networks that are initially identical to the Actor and Critic networks. The target networks further assist in dampening extreme behaviors by effectively taking a portion of the weight updates away from the main networks and applying them to their own weights, thereby dampening the learning process and smoothing the change in weights. The two Critic networks are updated by minimizing the mean squared temporal difference (TD) error loss, while the Actor network is updated by minimizing the negative mean Q-value under the current policy. Both updates use gradient descent. The algorithm is roughly provided below in Algorithm 1.

3. Guaranteeing stability for reinforcement learning-based control

The following section details the closed-loop form of the proposed RL control architecture alongside a formal derivation of its stability properties.

3.1. Closed-loop implementation

The closed-loop system operates with two main sources of control signals: the RL-controller and the fallback controller. The RL-controller is an NN trained using RL to function as a direct substitute for the LMPC, so it is capable of taking the sensor readings and providing approximately optimal control. The fallback controller is a short-horizon form of the LMPC that is only ever used if the constraint enforcer denies the RL-controller's solution. The constraint enforcer is an algorithmic check of the stability guarantee used in the LMPC design (i.e., either Eqs. (7e) or (8)). If the RL-controller violates this constraint, the fallback controller is used instead. Additionally, the constraint enforcer can enforce the control bounds by simply clipping the control signal within these bounds. As a final check, if the fallback control violates the constraint due to the LMPC solution failing, a failsafe controller is applied in the form of the reference controller used in Section 2.3. A block diagram representation of this system is shown in Fig. 1,

Algorithm 1 TD3

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$
Initialize replay buffer $\mathcal{D}$
while not converged **do**
  Observe the environment's current state s
  Select action with exploration noise $a = \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, 1)$
  Clip action within lower and upper bounds $a_{Lower} \leq a \leq a_{Upper}$
  Apply a and observe reward r, new state s', and done signal d
  Store transition tuple (s, a, r, s', d) in $\mathcal{D}$
  if `update_after` steps have been taken **and** step counter%`update_every` $= 0$ **then**
    Sample mini-batch of N transitions $(s_j, a_j, r_j, s'_j, d_j)_{j=1}^{N} \sim \mathcal{D}$
    $a'_j = \pi_{\phi'}(s'_j) + \epsilon_j$, $\epsilon_j \sim \text{clip}(\mathcal{N}(0, \texttt{policy\_noise}), -\texttt{noise\_clip}, \texttt{noise\_clip})$
    Clip $\tilde{a}'_j$ within lower and upper bounds $a_{Lower} \leq a' \leq a_{Upper}$
    $y_j = r_j + \gamma(1 - d_j)\min(Q_{\theta'_1}(s'_j, a'_j), Q_{\theta'_2}(s'_j, a'_j))$
    Compute critic losses: $\mathcal{L}_1 = \text{MSE}(Q_{\theta_1}(s_j, a_j), y_j)$, $\mathcal{L}_2 = \text{MSE}(Q_{\theta_2}(s_j, a_j), y_j)$
    $\mathcal{L}_{critic} = \mathcal{L}_1 + \mathcal{L}_2$
    Zero gradients for critics
    Backward pass on $\mathcal{L}_{critic}$
    Gradient descent step for θ_1 and θ_2
    if iteration counter $\bmod$ `policy_freq` $= 0$ **then**
      Compute actor loss: $\mathcal{L}_{actor} = -\frac{1}{N}\sum Q_{\theta_1}(s_j, \pi_\phi(s_j))$
      Zero gradients for actor
      Backward pass on $\mathcal{L}_{actor}$
      Gradient descent step for ϕ
      Soft update targets:
      $\theta'_k \leftarrow \tau\theta_k + (1 - \tau)\theta'_k$ for $k = 1, 2$
      $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
    end if
  end if
  if s' is in the terminal region **then**
    Reset environment
    Select a new initial state s
  end if
end while

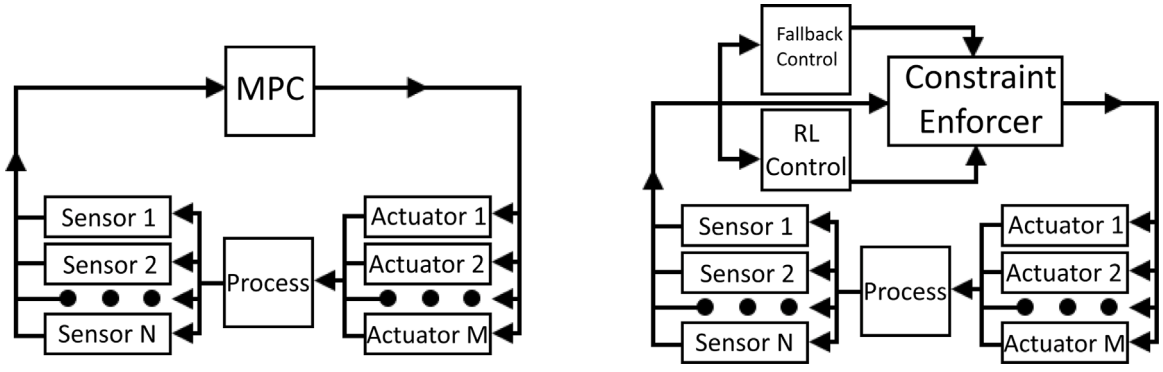


Fig. 1. Generalized process control block diagram for Model Predictive Control (Left) and Reinforcement Learning-based control (Right). The fallback control block contains a failsafe controller.

where all relevant controllers function as explicit feedback control laws, thereby using an RL-controller of the form:

$$\Phi_{\text{RL}} : \mathbb{R}^n \to \mathbb{R}^m \tag{22}$$

$$u(t_k) := \Phi_{\text{RL}}(x(t_k)) \tag{23}$$

Remark 7. The fallback controller does not need to be a short-horizon form of the LMPC. It can be any such controller that satisfies Eqs. (7e) or (8). A short-horizon LMPC is used as the fallback because it satisfies these constraints by design and additionally provides some form of cost optimization that simpler designs, such as P control, do not consider.

Remark 8. In order for a signal to pass through the constraint enforcer, it must satisfy all required constraints. Consider the case of requiring both clipping of the control signal to fit within the actuator bounds (the maximum magnitude of the respective control action) and enforcement of the Lyapunov stability constraint. It is important to note that in the constraint enforcer, the Lyapunov stability constraints are only checked after clipping is done. This is because the Lyapunov stability constraint is something that must be satisfied for the finalized (the one to be implemented on the process) control signal, whereas all control actions need to be clipped within the bounds before they can be considered finalized. Clipping is necessary to satisfy the control bounds if the model does not inherently satisfy the bounds by using a tanh

activation for its output layer. The control bounds exist as physical limits for a given actuator; therefore, they must be satisfied at all times. The Lyapunov stability constraint can be satisfied for a range of valid control actions, some of which may exceed the control bounds. But in order for a signal to pass the constraint enforcer, it must satisfy both the control bounds and the Lyapunov stability constraint, so it is possible for a control action to satisfy the stability constraint prior to clipping, but fail to satisfy it after clipping, thereby requiring fallback control that would then guarantee satisfaction of both.

Remark 9. The novelty of this framework is the ability to guarantee closed-loop stability through external constraint enforcement. The LMPC design, beyond the needed stability constraints, is not important to the framework's novelty, but it remains important to the particular design problems engineers will need to consider for their particular use cases.

Remark 10. RL is used for the RL-based controllers as a means of speeding up the training process. Imitation-learning-based methods face scaling issues in regards to the time it takes to solve the LMPC optimization problem for sufficiently diverse datasets. This is due to the increasing computational demands of LMPC for higher-dimensional problems and due to the increasing difficulty in densely and diversely sampling the relevant states. RL, through simulated interactions with a first-principles model of the process (prior to the real-time implementation of the RL-based controllers on the process), bypasses the scaling issues of the LMPC optimization problem entirely.

Remark 11. For both of the RL-based controllers and the FNN-based controller, training is done prior to real-time implementation of the controller on the process. In other words, all NN training in this work is done offline. While online training is possible through the use of transfer learning, this is not explored in the present work. The real-time implementation of these controllers to the process assumes that training is complete and that the constraint enforcer is present (to ensure closed-loop stability).

3.2. Closed-loop stability

For a continuous-time system, Section 2.3 provides assumptions that would guarantee exponential stability of the origin; however, continuous-time control is not possible due to the need for computation of control actions as well as signal transmission delays. To resolve this, the sample-and-hold control implementation applies a fixed control action for a fixed interval $t \in [t_k, t_k + \Delta)$ before updating. The consequence of this is that the stabilizability assumption guarantees convergence to a small region around the origin instead of exponential stability.

3.2.1. Closed-loop stability under LMPC

Theorem 1. *Take an LMPC as defined in Eq. (7), which applies either Eqs. (7e) or (8), and where Eq. (7b) models a system matching Eq. (1). $u(t)$ denotes the first control input vector from the LMPC's solution. If a reference controller and Lyapunov function described by Section 2.3 exists for $x(t_0) \in D := \{x \mid V(x) \le \rho\}$, then there must exist a pair of constants $\epsilon_w, \alpha > 0$ such that the closed-loop state is driven to a small area around the origin ($\Omega_{\rho_{\min}}$) by the sample-and-hold implementation of $u(t)$ if the conditions below hold for any sampling instant t_k:*

$$L_x' M_F \Delta - \alpha \rho_s \le -\epsilon_w \tag{24a}$$

$$\rho_{\min} := \max\left\{V\left(x\left(t_k + \Delta\right) | V\left(x\left(t_k\right)\right) \le \rho_s\right)\right\} \tag{24b}$$

$$\rho_s < V\left(x\left(t_k\right)\right) \tag{24c}$$

$$\rho_s < \rho_{\min} < \rho \tag{24d}$$

Proof.

Case 1 (*Eq. (7e) as the Lyapunov constraint*)**.** *Starting with the definition of the time derivative of the Lyapunov function, we can manipulate the expression algebraically by adding and subtracting equivalent terms:*

$$\dot{V}\left(x(t), u\left(t_k\right)\right) = \frac{\partial V(x(t))}{\partial x} F\left(x(t), u\left(t_k\right)\right) \tag{25}$$

$$\begin{aligned}\dot{V}\left(x(t), u\left(t_k\right)\right) = &\frac{\partial V(x(t))}{\partial x} F\left(x(t), u\left(t_k\right)\right) \\ &- \frac{\partial V\left(x\left(t_k\right)\right)}{\partial x} F\left(x\left(t_k\right), u\left(t_k\right)\right) \\ &+ \frac{\partial V\left(x\left(t_k\right)\right)}{\partial x} F\left(x\left(t_k\right), u\left(t_k\right)\right)\end{aligned} \tag{26}$$

Plugging in Eqs. (5b), (6c) and (7e):

$$\dot{V}\left(x(t), u\left(t_k\right)\right) \le L_x' \left|x(t) - x\left(t_k\right)\right| - c_3 \left|x\left(t_k\right)\right|^2 \tag{27}$$

Via the integral triangle inequality:

$$\left|x(t) - x\left(t_k\right)\right| \le \int_{t_k}^{t} \left|F\left(x(\tau), u\left(t_k\right)\right)\right| \mathrm{d}\tau \le M_F \Delta \tag{28}$$

Plugging into Eq. (27):

$$\dot{V}\left(x(t), u\left(t_k\right)\right) \le L_x' M_F \Delta - c_3 \left|x\left(t_k\right)\right|^2 \tag{29}$$

If $|x|$ is small, the $L_x' M_F \Delta$ term may dominate and make the upper bound of $\dot{V}$ positive, invalidating any stability guarantee; hence the need for Eq. (24c). Applying this with Eq. (5a) yields:

$$\dot{V}\left(x(t), u\left(t_k\right)\right) \le L_x' M_F \Delta - \frac{c_3}{c_2} \rho_s \le L_x' M_F \Delta - \alpha \rho_s \tag{30}$$

which simplifies using Eq. (24a):

$$\dot{V}\left(x(t), u\left(t_k\right)\right) \le -\epsilon_w \tag{31}$$

Thus, with sufficiently small Δ, any closed-loop state $x\left(t_k\right) \in \Omega_\rho \backslash \Omega_{\rho_s}$ will decay over time towards Ω_{ρ_s}, ultimately converging to $\Omega_{\rho_{\min}}$.

Case 2 (*Eq. (8) as the Lyapunov constraint*)**.** *The proof for this cases follows the same form as above with the exception of using Eq. (8) as opposed to Eq. (7e), which results in the $-c_3 \left|x\left(t_k\right)\right|^2$ term being replaced by $-\alpha V\left(x\left(t_k\right)\right)$.*

$$\begin{aligned}\dot{V}\left(x(t), u\left(t_k\right)\right) &\le L_x' \left|x(t) - x\left(t_k\right)\right| - \alpha V\left(x\left(t_k\right)\right) \\ &\le L_x' M_F \Delta - \alpha \rho_s\end{aligned} \tag{32}$$

Remark 12. The alternate form of the stability constraint from Eq. (8) does not explicitly use the reference controller, but still requires it to exist due to the Lyapunov function needing to satisfy Eq. (5). A consequence of using this form is that α must be defined by the user. An excessively small α risks being overpowered by the $L_x' M_F \Delta$ term, whereas an excessively large α risks the solution being infeasible due to the control bounds.

Remark 13. The proof demonstrates stability guarantees for any such interval in which the stability constraints are enforced. As presented in Eq. (7), this implies that stability guarantees do not exist beyond the first sampling interval; hence, the receding horizon approach would functionally satisfy the stability guarantees but is not guaranteed to optimize with respect to a trajectory that satisfies these guarantees for all points beyond the first sampling interval.

3.2.2. Closed-loop stability under RL-controller with constraint enforcement

As seen in Fig. 1, the framework operates by conditionally selecting which control signal to send to the actuators. Despite the RL-controller having no stability guarantees, the stability guarantees for the LMPC shown in Theorem 1 also guarantee stability for the modified framework via the constraint enforcer.

Theorem 2. *Consider an LMPC as given in Eq. (7), which employs either Eq. (7e) or Eq. (8) as the stabilizability constraint, and where Eq. (7b) represents a nonlinear system matching the structure in Eq. (1). Assume a reference controller and Lyapunov function described by Section 2.3 exists for* $x(t_0) \in D := \{x \mid V(x) \leq \rho\}$, *and a reinforcement learning-based controller with control outputs clipped to meet the constraints in Eq. (7c)* ($\Phi_{\mathrm{RL}}(x(t_k)) \in U$) *exists. For the RL-based process setup illustrated in Fig. 1 whose constraint enforcer is defined as*

$$u(t_k) = \begin{cases} \Phi_{\mathrm{RL}}(x(t_k)), & \text{if } \dot{V}(x(t_k), \Phi_{\mathrm{RL}}(x(t_k))) \leq S(x(t_k)) \\ \Phi_{\mathrm{LMPC}}(t_k), & \text{if } \dot{V}(x(t_k), \Phi_{\mathrm{RL}}(x(t_k))) > S(x(t_k)) \end{cases} \tag{33}$$

where $S(x)$ *denotes the stability threshold—defined as either* $\dot{V}(x, \Phi(x))$ *or* $-\alpha V(x)$ *depending on the chosen form of the constraint—and*

$$\Phi_{\mathrm{LMPC}}(t_k) = \begin{cases} u_{\mathrm{LMPC}}(t_k), & \text{if } \dot{V}(x(t_k), u_{\mathrm{LMPC}}(t_k)) \leq S(x(t_k)) \\ \Phi(x(t_k)), & \text{if } \dot{V}(x(t_k), u_{\mathrm{LMPC}}(t_k)) > S(x(t_k)) \end{cases} \tag{34}$$

such that $u, u_{LMPC}, \Phi, \Phi_{\mathrm{RL}}, \Phi_{LMPC} \in U$, *then there must exist a pair of constants* $\epsilon_w, \alpha > 0$ *such that the closed-loop state is driven to a small area around the origin* ($\Omega_{\rho_{\min}}$) *by the sample-and-hold implementation of* $u(t)$ *if the conditions in Eq. (24a)–(24d) hold for any sampling instant* t_k

Proof.

Case 1 ($\dot{V}(x(t_k), \Phi_{\mathrm{RL}}(x(t_k))) > S(x(t_k))$). *We consider the remaining two subcases for the LMPC control action.*

Case 1.1 ($\dot{V}(x(t_k), u_{\mathrm{LMPC}}(t_k)) > S(x(t_k))$). *Here,* $\Phi_{LMPC}(t_k)$ *is used and equals* $\Phi(x(t_k))$. *The proof follows Theorem 1, but Eq. (26) is composed of* $\Phi(x)$ *instead. Because the Lipschitz continuity expression from Eq. (6c) is applicable for any u, and Eq. (5b) is satisfied, the case simplifies to Case 1 from Theorem 1.*

Case 1.2 ($\dot{V}(x(t_k), u_{\mathrm{LMPC}}(t_k)) \leq S(x(t_k))$). *Here,* $\Phi_{LMPC}(t_k)$ *is used and equals* $u_{LMPC}(t_k)$, *the first control action of the LMPC solution. Theorem 1 details the proof for this controller.*

Case 2 ($\dot{V}(x(t_k), \Phi_{\mathrm{RL}}(x(t_k))) \leq S(x(t_k))$). *This case follows the steps from Theorem 1, but Eq. (26) is composed of* $\Phi_{\mathrm{RL}}(x)$ *terms instead. The Lipschitz continuity expression from Eq. (6c) is applicable for any u, thus the case simplifies to the form used in Theorem 1 where the respective case is chosen depending on the form of* $S(x)$.

Remark 14. The RL-controller has no stability guarantees alone, hence why every case eventually reforms to be in terms of the reference controller, as the reference controller's existence and use are solely for the enforcement and satisfaction of stability guarantees.

4. Application to a chemical process example

In this section, the proposed stable RL framework is implemented on a chemical process, and the efficacy of the framework is demonstrated by comparing it with various controllers.

4.1. Process description

The model chemical process of choice for this study is a simulated continuous stirred-tank reactor (CSTR). The CSTR is assumed to be perfectly mixed. Specifically, we consider a singular irreversible reaction that is exothermic, making the CSTR non-isothermal. The reaction is treated as an arbitrary liquid-phase reaction ($A \rightarrow B$) with second-order dynamics. The CSTR is insulated in the sense that there is no external influence on the system's heat, but heat is still removed or added to the system through a heating/cooling system implemented on the reactor, with heat addition or removal occurring at a controllable rate $\dot{Q}$. These assumptions yield the following dynamic model:

$$\frac{\mathrm{d}C_A}{\mathrm{d}t} = \frac{F}{V_L}(C_{A0} - C_A) - kC_A^2 \tag{35a}$$

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{F}{V_L}(T_0 - T) - \frac{\Delta H}{\rho_L C_p} kC_A^2 + \frac{\dot{Q}}{\rho_L C_p V_L} \tag{35b}$$

$$k = k_0 \exp\left[-\frac{E}{RT}\right] \tag{35c}$$

Here—with the exception of k_0, which denotes the isothermal rate-constant—the 0 subscript denotes feed values, C_A denotes concentration of A, and T denotes the temperature of the solution within the CSTR. $\rho_L, C_p, \Delta H, E$ and V_L denote the solution density, specific heat, heat of reaction, activation energy, and liquid volume, respectively.

Table 1
Parameter values of the CSTR model.

Var.	Value	Var.	Value
C_{As}	$1.954\,\mathrm{kmol\,m^{-3}}$	C_{A0s}	$4.0\,\mathrm{kmol\,m^{-3}}$
C_p	$0.231\,\mathrm{kJ\,kg^{-1}\,K^{-1}}$	ΔH	$-11{,}500\,\mathrm{kJ\,kmol^{-1}}$
E	$5.0 \times 10^4\,\mathrm{kJ\,kmol^{-1}}$	F	$10\,\mathrm{m^3\,h^{-1}}$
k_0	$8.46 \times 10^6\,\mathrm{m^3\,kmol^{-1}\,h^{-1}}$	$\dot{Q}_s$	$0.0\,\mathrm{kJ\,h^{-1}}$
R	$8.314\,\mathrm{kJ\,kmol^{-1}\,K^{-1}}$	ρ_L	$1.0 \times 10^3\,\mathrm{kg\,m^{-3}}$
T_0	300.0 K	T_{0s}	300.0 K
T_s	401.9 K	V_L	$0.1\,\mathrm{m^3}$

4.2. Control problem

The heating rate $\dot{Q}$ is selected as the control (manipulated) input, and the state variables are chosen to be T and C_A. In order to utilize the origin as the steady state (denoted by the s subscript) without loss of generality, the state and control variables are expressed as deviation variables. Accordingly, the vectors used in the problem formulation are defined as $x^\top = [C_A - C_{As}, T - T_s]$ and $u^\top = [\dot{Q} - \dot{Q}_s]$ for the state and control vectors, respectively. The control input is subject to bounds, specifically $-4{,}000 \leq \dot{Q} - \dot{Q}_s \leq 4{,}000\,\mathrm{kJ\,h^{-1}}$. Specifics on the various constants used in the CSTR dynamic model are provided in Table 1.

The design goal is to create a controller that drives the closed-loop system from any given initial state bounded by $-0.6 \leq C_A - C_{As} \leq 0.6\,\mathrm{kmol\,m^{-3}}$ and $-10 \leq T - T_s \leq 10\,\mathrm{K}$ to the origin. Because of the deviation variable notation, this origin represents the desired operating (unstable) steady state. To achieve this, a reference controller satisfying Section 2.3 is found to be a proportional (P) controller with a weight of 120 for the temperature deviation variable. Similarly, a Lyapunov function of the form $V = x^\top P x$ with $P = \begin{bmatrix} 2{,}033 & -0.00051 \\ -0.00051 & 0.00070 \end{bmatrix}$ is used.

The immediate cost of the LMPC and RL at $t = t_k$ is designed as follows:

$$L(T(t_k), \dot{Q}(t_k)) = a(T(t_k) - T_s)^2 + b(\dot{Q}(t_k) - \dot{Q}_s)^2 \tag{36}$$

where a and b are the weight coefficients for the state variable ($T(t_k)$) and control input ($\dot{Q}(t_k)$), respectively. In particular, $a = 1$ and $b = 6{\times}10^{-7}$. The consistent cost function is used to ensure a fair comparison between all controllers.

4.3. LMPC design

The system operates with a sampling time of $\Delta = 1\,\mathrm{s}$. Two LMPC formulations are utilized: a long-horizon LMPC with horizon length $N = 50$ and a short-horizon LMPC with horizon length $N = 5$. The sole distinction lies in their prediction windows: the long-horizon LMPC optimizes trajectories over 50 s, whereas the short-horizon LMPC

optimizes over only 5 s. While the long-horizon design achieves better cost optimization, making it ideal as a reference for near-optimal control, its computational demand causes the solution time to exceed the sampling period, making it unsuitable for real-time implementation. In contrast, the short-horizon LMPC has a smaller computational demand, which allows it to solve within the sampling period's time frame. Consequently, the short-horizon LMPC can be deployed for real-time closed-loop implementations at the expense of poorer cost optimality.

In this work, the optimization problem of the LMPC is solved with two different methods. The LMPC solution used for the HJB-based RL network example utilizes the sequential least squares quadratic programming (SLSQP) algorithm, a gradient-based method designed for constrained nonlinear programs. It approximates the nonlinear problem with a quadratic subproblem at each step and updates iteratively. After running test simulations to determine optimal parameters, the convergence tolerance and finite-difference step size are set as 1×10^{-10} and 1×10^{-5}. The LMPC solution used for the TD3-based RL network example utilizes IPOPT with the SPRAL linear solver. This was done using the do-MPC Python package with further customizations enabled as follows:

- `max_iter`: 10,000
- `tol`: 1×10^{-8}
- `acceptable_tol`: 1×10^{-6}
- `mu_strategy`: 'adaptive'
- `warm_start_init_point`: 'yes'
- `expect_infeasible_problem`: 'yes'

The do-MPC package also provides options for how the process is simulated forward in time (Fiedler et al., 2023). This was designed to be a collocation-type integrator using the `radau` method with 5 degrees of collocation over a total step size of Δ.

Remark 15. Two different implementations of the LMPC framework are used in this work. Although the implementations differ on a technical level, the fundamental formulation and functional difference between the two methods are negligible. The implementation of the LMPC framework does not need to mimic the methods shown in this work, as any numerically valid implementation of LMPC that satisfies the fundamental formulation is valid. Both methods shown use precise enough tolerances and numerical integration to have roughly the same solution for a given state, with the exception of cases where SLSQP or IPOPT fail to solve, which is a possibility for some regions in the state-space.

Remark 16. The long-horizon LMPC is primarily used as a reference for what is effectively the optimal control logic. Because of this, it is used offline to generate data that can then be used for imitation-learning-based pre-training. The short-horizon LMPC is used as the back-up controller during real-time implementation of the various controller designs due to its stability guarantees and improved cost over the reference stabilizing controller.

4.4. FNN-based control design

To address the limitations associated with both short-horizon and long-horizon LMPC, an FNN-based control framework was recently proposed to approximate the functionality of the long-horizon LMPC while maintaining fast computational performance (Khodaverdian et al., 2025b, 2026). In this work, the FNN-based control implementation is used for relative comparison with the performance of RL-based control. Specifically, 10,000 initial state vectors are randomly sampled from the feasible domain and treated as the system states at $t = 0$. From each initial condition, closed-loop simulations are performed using the long-horizon LMPC until $t = 4$ min, and the resulting state and control trajectories are recorded to form the training dataset for the neural network.

Table 2
Best hyperparameters of FNN obtained from Bayesian optimization.

Hyperparameter	Value	Hyperparameter	Value
Learning rate	0.000148	Batch size	32
Epochs	61	Width	410
Depth	4	Dropout	0.051

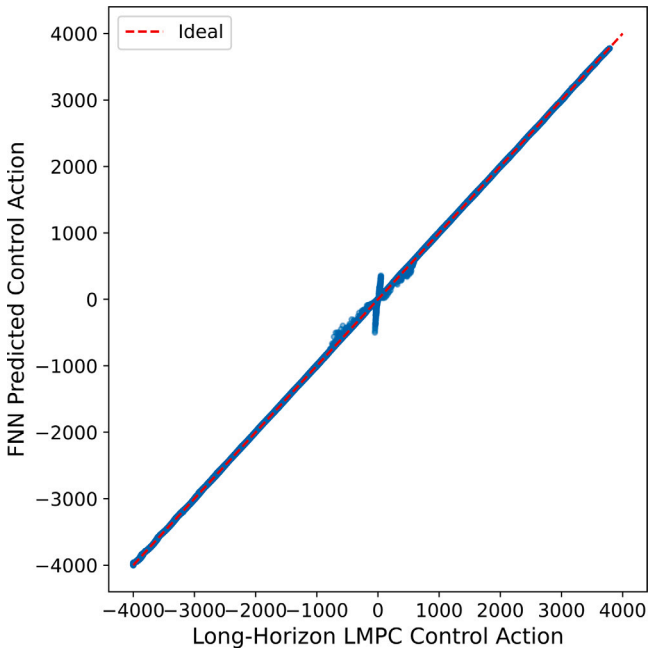


Fig. 2. Comparison of the trained FNN control actions to the desired long-horizon LMPC control actions.

During the training process, the network takes the normalized states as input and outputs the scaled action u through a tanh activation. The hidden layers use ReLU nonlinearities, and dropout regularization is applied to mitigate overfitting. To select the network architecture and training settings, a Bayesian optimization procedure (Gaussian process minimization) was carried out over the hyperparameter space, including learning rate, batch size, training epochs, hidden width, hidden depth, and dropout rate. The search was performed using 30 evaluations, with validation loss as the objective function. The resulting optimal hyperparameters are reported in Table 2.

These values were subsequently employed to train the final actor network, which was then evaluated on the test set. Fig. 2 illustrates the ability of the FNN model to reproduce the long-horizon LMPC behavior, as the data points lie closely along the ideal line $y = x$.

Remark 17. Although the long-horizon LMPC is designed with a sampling time of 1 s, its per-step computation time exceeds this duration. This is acceptable for offline data generation but problematic in real-time implementation, where delays between measurements and control actions are critical.

Remark 18. The FNN-based controller primarily functions as a baseline for comparison between other NN-based controllers. Although it can also serve as a valid controller in this framework, the imitation-learning-based training process limits the design to applications where there exists sufficient data from the LMPC.

4.5. HJB-based RL-controller design

The HJB-based RL-controller is designed using the CSTR model. In particular, the cost function is $L = a(T - T_s)^2 + b(\dot{Q} - \dot{Q}_s)^2$, which is

Table 3
RL training hyperparameters (HJB value-critic).

Hyperparameter	Value	Hyperparameter	Value
Optimizer	Adam	Learning rate	0.001
Batch size	256	Exploration noise	0.10
Eval frequency	960	Start steps	960
Max steps	10,000	Buffer size	1,000,000
Net width (units)	128	Net depth (layers)	2

the same expression as the LMPC cost function. The associated analytic policy derived from Eq. (20) is

$$\Phi_{RL}(x) = \text{clip}\left(-\frac{\partial V_w(x)/\partial T}{2bV_L\rho_L C_p}, \Phi_{\min}, \Phi_{\max}\right) \tag{37}$$

where the $\Phi_{\min} = -4{,}000\,\text{kJ}\,\text{h}^{-1}$ and $\Phi_{\max} = 4{,}000\,\text{kJ}\,\text{h}^{-1}$. During the training process, we first sample the initial condition randomly on deviation states within the feasible region. From each sampled initial condition, we generate training data by simulating a 4-minute trajectory. This trajectory data is normalized by min–max scaling to $[0, 1]$ and internally rescaled back to physical units before computing $\partial V_w/\partial T$ in Eq. (37). The value network V_w is trained on physical deviation states. The value network V_w is trained by minimizing the squared HJB residual (Eq. (16)) optimized with Adam.

Transition data (s, a, s', r) are stored in a replay buffer with a capacity of 1×10^6, where r denotes the instant cost calculated based on s' and a (the same as the LMPC cost). HJB updates are done with a mini-batch of typical size 256 drawn from a replay buffer that is populated by on-policy rollouts with small Gaussian action noise. In order to allow the buffer to grow to a sufficient size before sampling, for the first 960 steps, we draw samples uniformly from the feasible region. Using random Gaussian actions and the corresponding r and s', the replay buffer is initially padded with data that does not involve the policy. After this initial random-action phase, on-policy rollouts use small Gaussian action noise with standard deviation 0.10 times the actuation limit (the maximum magnitude of the respective control action) to encourage exploration. Evaluation is performed every 960 steps on fresh episodes, and evaluation scores are logged for subsequent visualization.

Evaluation follows the same procedure as the real implementation. After computing the control action from RL, if the Lyapunov derivative under the RL policy is larger than that under the P controller (i.e., $\dot{V}(x, \Phi_{\text{RL}}(x)) > \dot{V}(x, \Phi(x))$), we solve a short-horizon LMPC problem and apply Φ_{LMPC}; otherwise, we apply Φ_{RL}. To prevent performance drift during training, we employ a best-so-far acceptance gate: at fixed evaluation intervals, candidate parameters are tested on fresh episodes and accepted only if the average return improves; otherwise, we revert to the last committed w and reset the optimizer state. This procedure ensures that the trained RL policy does not degrade relative to its previous version under the chosen evaluation protocol. The detailed hyperparameters are reported in Table 3.

4.6. TD3-based RL-controller design

The TD3-based RL-controller uses the same CSTR system as the HJB-based RL-controller; however, the LMPC aims to minimize the cost over the horizon, while TD3 aims to maximize the reward function. Additionally, the TD3 algorithm uses discrete values while the LMPC is defined for continuous-time systems. Since the controller is applied in a sample-and-hold fashion and the state is measured in fixed sampling time intervals, the LMPC problem can be viewed in discrete intervals based on the sampling time. To approximate the cost, the integral cost function is replaced with a sum of the quadratic cost terms. Due to this discrete form, the control action will be the value calculated for the reference time frame t_k, whereas the state will be the approximate future state at a time $t_k + \Delta$. In other words, we can represent the reward function as

$$r(s', a) = -\left(s'^{\top} W_x s' + a W_u a\right) \tag{38}$$

The TD3 algorithm has the benefit of being both model-free and off-policy, which leads it to use the action-value function instead of the state-value function. Additionally, the TD3 method is an Actor–Critic method, meaning that the policy is explicitly expressed as a neural network that is iteratively improved during training instead of being analytically solved using the value function, as is done in the HJB-based approach.

While TD3's strength lies in the various differences mentioned above, effectively tuning its hyperparameters requires a method of quantifying the Actor and Critic's quality. This can be somewhat done by observing the temporal difference (TD) error and the actor-value function itself to ensure that both converge to a stable point; however, the algorithm is inherently noisy, which makes it difficult to truly determine if these values have converged. As such, the primary mode of evaluation was to simulate the closed-loop system using the TD3-based controller without fallback control over many initial points. The cost of the resulting trajectory over a sufficiently long horizon is then compared with the cost of the long-horizon LMPC starting at the same initial point. The logic behind this evaluation criterion is that the percent error relative to the long-horizon LMPC would be a good indicator of model performance if done for enough samples. Additionally, this consolidates the convergence criteria to a single parameter.

Furthermore, to mitigate the initial variance and instability of the TD3 algorithm, there is a pretraining step for both the Actor and Critic networks. Using `AdamW` as the optimizer and `OneCycleLR` as the scheduler with the `max_lr` being set equal to the corresponding learning rate, the Actor and Critic undergo separate pre-training loops. The Actor is pretrained with behavior cloning (BC) via mean squared error loss relative to pre-generated data from the LMPC system's trajectories. The trajectories are based on randomly sampled initial points within the operating region of the system. This dataset is used to pretrain the Actor over 10 epochs, each of which covers the entire dataset once. The same dataset is used to generate the `StandardScaler` with the goal of improving the model's ability to learn by scaling the Actor's inputs to have zero mean and unit variance. The Critics are trained using the same loss function as the TD3 algorithm, as defined in steps 14 and 15 in Algorithm 1, but the LMPC data, formatted as (s, a, r, s', a', d), is used. Thus, no policy noise is applied during pretraining, and the LMPC's dataset is treated as the replay buffer with the future action being explicitly stored. In other words, the a' is no longer calculated using the target policy with noise, and is instead stored exactly using the LMPC's dataset. This modified form of the replay buffer is referred to as the formatted LMPC dataset. The Critics are pre-trained over 5 epochs, each of which covers the entire formatted LMPC dataset once. After the Actor and both Critics are pretrained, the target networks are finally initialized using the pretrained parameters of their corresponding networks.

The TD3 method's exploration steps are useful for exploring the state-space, but excessive exploration is undesirable later in training. To account for this, every step of exploration in the environment decays the exploration noise by a fixed percent until a minimum is hit. Combined with the other modification to the TD3 method described above, we define the modified training loop as its own algorithm described in Algorithm 2. These modifications, along with the existing complexity of the TD3 algorithm, introduce a large amount of hyperparameters that can be seen as important. As such, the controller must go through two distinct training loops. First, the controller will undergo hyperparameter tuning using Optuna (Akiba et al., 2019). Then, the best performing hyperparameters will be used in an extended training loop to allow for further completion of the modified TD3 tuning. The

Algorithm 2 Modified TD3 with BC Pretraining and Exploration Noise Decay

Initialize critic networks Q_{θ_1}, Q_{θ_2}, and actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Pretrain Actor via BC: Optimize π_ϕ on expert states s_j and actions a_j from LMPC dataset using MSE loss $\mathcal{L}_{BC} = \text{MSE}(\pi_\phi(s_j), a_j)$ for 10 epochs
Pretrain Critics via BC: Optimize Q_{θ_1}, Q_{θ_2} on expert transitions $\left(s_j, a_j, r_j, s'_j, a'_j, d_j\right)$ using TD targets $y_j = r_j + \gamma\left(1 - d_j\right)\min\left(Q_{\theta'_1}\left(s'_j, a'_j\right), Q_{\theta'_2}\left(s'_j, a'_j\right)\right)$ for 5 epochs
Initialize target networks $\theta'_1 \leftarrow \theta_1$, $\theta'_2 \leftarrow \theta_2$, $\phi' \leftarrow \phi$ and replay buffer $\mathcal{D}$
Initialize environment
while less than max steps taken **and** not early stopped on instability **do**
  Decay exploration noise $\sigma_{expl} \leftarrow \max\left(\texttt{min\_expl\_noise}, \sigma_{expl} \cdot \texttt{expl\_decay\_rate}\right)$
  if not enough policy warmup steps taken **then**
    Select random action $a \sim \mathcal{U}(a_{Lower}, a_{Upper})$
  else
    $a = \pi_\phi(s) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma_{expl})$, Clip action $a_{Lower} \le a \le a_{Upper}$
  end if
  Apply a and observe reward r, new state s', and done signals d^* (s' converged or max steps reached), d (s' converged)
  Store transition tuple (s, a, r, s', d) in $\mathcal{D}$ and increment the environment step counter
  if d^* **then**
    Reset environment and select a new initial state s
  end if
  if replay buffer is sufficiently large **and** environment steps%steps per critic update **then**
    Sample mini-batch of N transitions $(s_j, a_j, r_j, s'_j, d_j)_{j=1}^N \sim \mathcal{D}$
    $a'_j = \pi_{\phi'}(s'_j) + \epsilon_j$, $\epsilon_j \sim \text{clip}(\mathcal{N}(0, \texttt{policy\_noise}), -\texttt{noise\_clip}, \texttt{noise\_clip})$
    Clip action $a_{Lower} \le a' \le a_{Upper}$
    Compute $y_j = r_j + \gamma(1 - d_j)\min(Q_{\theta'_1}(s'_j, \tilde{a}'_j), Q_{\theta'_2}(s'_j, \tilde{a}'_j))$
    Compute critic losses: $\mathcal{L}_1 = \text{MSE}(Q_{\theta_1}(s_j, a_j), y_j)$, $\mathcal{L}_2 = \text{MSE}(Q_{\theta_2}(s_j, a_j), y_j)$
    Gradient descent step for θ_1 and θ_2 with critic learning rate for $\mathcal{L}_{critic} = \mathcal{L}_1 + \mathcal{L}_2$
    Increment total step counter
    if total steps%steps per policy update **then**
      Gradient descent step for ϕ using $\mathcal{L}_{actor} = -\frac{1}{N}\sum Q_{\theta_1}(s_j, \pi_\phi(s_j))$
      Soft update targets:
      $\theta'_k \leftarrow \tau\theta_k + (1 - \tau)\theta'_k$ for $k = 1, 2$
      $\phi' \leftarrow \tau\phi + (1 - \tau)\phi'$
    end if
  end if
  if environment steps%steps per evaluation **then**
    Evaluate policy on multiple trajectories, compute %error vs. LMPC
    Compute average TD error over entire replay buffer
    if std of recent %errors <`rel_error_threshold` and std of recent TD errors<`td_threshold` for last 10 evaluations **then**
      Mark as converged, save checkpoint
    else if no improvement in %error for `patience` evals **or** Optuna wishes to prune **then**
      Prune trial
    end if
  end if
end while

resulting controller is then applied to the process for 100 samples for analysis.

The Optuna hyperparameter tuning is set up as a robust sweep of possible options for the modified TD3 system. The tuned parameters, their ranges, and the scaling type are as shown in Table 4. This Optuna study was set to run indefinitely until user termination, with the goal of minimizing the percent error of the average trajectory cost of the RL-controller vs a long-horizon LMPC-controller baseline. Using the training algorithm shown in Algorithm 2, and the constant hyperparameters displayed in Table 6, the best trial out of 110 samples was found to have the hyperparameters displayed in Table 5. This model was then run through the full training cycle, which increased the max steps to 10 million, patience to 100, maximum steps per episode to 500, and evaluation frequency to 25.000 (`start_steps` and `update_after` are set equal to this new value). Unlike the HJB-based RL-controller, this model did not go through a robust testing process. Instead, the TD3-based RL-controller was applied in a closed-loop fashion to the system, starting at 100 random initial points within a bounded state range.

Table 4
Optuna tuned parameter ranges and scaling for the TD3-based RL-controller.

Hyperparameter	Lower bound	Upper bound	Scale/Type
Actor Learning Rate	1×10^{-7}	1×10^{-2}	Log
Critic Learning Rate	1×10^{-7}	1×10^{-2}	Log
Actor Pretraining Learning Rate	1×10^{-7}	1×10^{-2}	Log
Soft Update Coefficient	0.0001	0.01	Log
Policy Noise	0.001	2	Float
Exploration Noise	0.001	0.2	Float
Hidden Layer Size	64	1,024	2^n
Number of Hidden Layers	1	6	Int
Steps per Policy Update	1	4	Int
Use LayerNorm	False	True	Bool

4.7. Closed-loop simulation results

4.7.1. HJB-based controller results

The closed-loop setpoint tracking behavior from the initial state $T - T_s = 10\,\text{K}$ and $C_A - C_{As} = 0.6\,\text{kmol m}^{-3}$ is illustrated in Fig. 3.

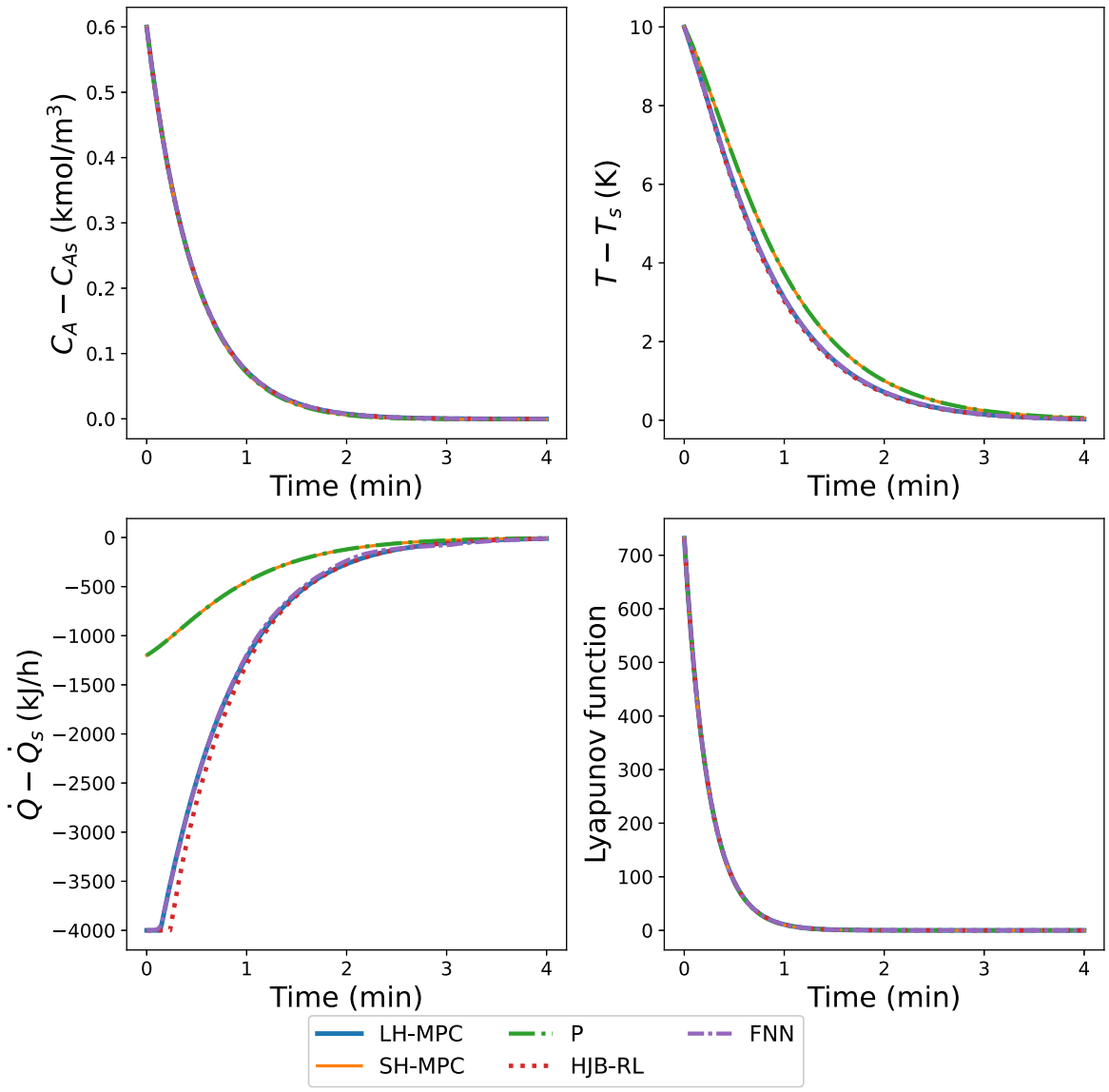


Fig. 3. Closed-loop trajectories of concentration deviation, temperature deviation, heat input, and Lyapunov function with the initial states at [10 K, 0.6 kmol m^{-3}]. Results are shown for five controllers: long-horizon LMPC (LH-LMPC), short-horizon LMPC (SH-LMPC), P controller (P), FNN-based approximate LMPC (FNN) and Hamilton–Jacobi–Bellman reinforcement learning (HJB-RL).

Compared with the long-horizon LMPC, the short-horizon LMPC applies smaller initial control actions due to its emphasis on immediate cost, similar to the P controller, whereas the long-horizon LMPC accounts for relatively long-term costs, and therefore issues larger initial actions. The FNN closely approximates the long-horizon LMPC and thus exhibits nearly identical behavior. In contrast, the RL controller, which optimizes an infinite-horizon cost, takes actions different from those of the LMPC, particularly in the initial region when the states are far from the setpoint. For all controllers, the Lyapunov function decreases monotonically over time, validating closed-loop stability.

To evaluate the controllers, a total of 500 initial states were generated using a stratified sampling strategy. The state-space was divided into 50 rectangular regions defined by evenly spaced intervals of the domain ($C_A - C_{As} \in [-0.6, 0.6]$ and $T - T_s \in [-10, 10]$). From each region, 10 initial states were uniformly sampled, resulting in 500 distinct starting points. Each sampled state was then controlled by every controller under identical simulation conditions, with a maximum simulation horizon of 4 min. The simulation was terminated early if the system entered a small neighborhood of the steady state (i.e., $|C_A - C_{As}| < 0.006$ and $|T - T_s| < 0.1$), ensuring that the performance metrics reflected the actual stabilization time. As summarized in Table 7 and Fig. 4, we take long-horizon LMPC as the 0.0 % baseline—delivering strong setpoint tracking but at huge computational cost (mean 2,181 ms, max 11,520 ms). Relative to this baseline, the proposed RL controller attains a slightly lower cost (−0.1 %) while reducing

Table 5
Best performing hyperparameters for the TD3-based RL-controller after Optuna tuning for 110 trials.

Variable (Hyperparameter)	Value
`lr`	$1.775275759804681 \times 10^{-7}$
`critic_lr`	0.0005089627893036739
`hidden_dim`	128
`num_hidden_layers`	5
`expl_noise_std`	0.13951586331641103
`tau`	0.007988601140325966
`policy_freq`	2
`bc_lr`	$4.8579319207288955 \times 10^{-5}$
`use_layernorm`	False
`policy_noise`	0.10712953866806144

Table 6
Constant hyperparameters for the TD3-based RL-controller during Optuna tuning.

Variable	Value	Description
buffer_size	2,000,000	Size of the replay buffer
batch_size	512	Mini-batch size for replay buffer sampling
gamma	0.99	Discount for future rewards
max_episode_steps	400	Max steps per episode
total_steps	100,000	Max environment interactions
start_steps	10,000	Initial random actions before policy use
update_after	512	Steps before starting updates
update_every	1	Update frequency after warm-up
weight_decay	10^{-5}	L2 regularization for optimizers
bc_pretrain_epochs	10	Epochs for actor BC pretraining
bc_weight_decay	10^{-5}	Weight decay for BC optimizer
expl_decay_rate	0.99975	Per-environment step decay for exploration noise
min_expl_noise	10^{-3}	Floor for exploration noise std
eval_freq	5000	Steps between evaluations
eval_episodes	10	Trajectories simulated per evaluation
converge_eps	0.01	State norm threshold for convergence
rel_error_threshold	5.0	Std threshold for actor stability (%error)
td_threshold	0.01	Std threshold for critic TD error stability
patience	10	Epochs of no improvement before early stop
stability_window	10	Recent evals for stability checks
num_trials	50	Optuna trials for tuning
critic_pretrain_epochs	5	Epochs for critic warm-up pretraining
noise_clip	2.5× policy_noise	Derived clip for target policy noise

Table 7
Relative costs of the proposed HJB-RL-based controller versus other controllers. Relative costs are normalized to LH-LMPC (0.0 %), with negative values indicating improvement.

	P Controller	SH-LMPC	LH-LMPC	FNN	HJB-RL
Cost (%)	4.8	4.8	0.0	0.0	−0.1

average computational time to 0.644 ms and worst-case time cost to 67.5 ms, comfortably within the 1 s sampling budget. A plain FNN policy matches the baseline cost (0.0 %) with modest computational burden (mean 2.32 ms, max 12.5 ms), but HJB-RL delivers the best setpoint performance with sub-millisecond average time. By contrast, short-horizon LMPC and P control both exhibit markedly higher costs (4.8 %); short-horizon LMPC also shows nontrivial computation (mean 15.247 ms, max 112.516 ms), whereas P control is computationally light but performance-limited. Overall, the proposed RL method provides the most balanced controller—combining the best closed-loop cost among all methods with practical real-time implementation under the one-second sampling period. That said, it is important to clarify that during HJB-RL or FNN implementation, the closed-loop stability enforcer checks at each sampling time, the Lyapunov function-based closed-loop stability constraint, and if this constraint is violated by the control action calculated by the HJB-RL or FNN, then the back-up stabilizing controller calculated control action is implemented. Therefore, no rigorous statement can be made that the improved closed-loop performance observed under HJB-RL is the result of implementing the HJB-RL most of the sampling times during this closed-loop simulation run. The expectation is that the HJB-RL would provide improved closed-loop performance over a back-up stabilizing controller but no a priori guarantee can be made about such an outcome.

A further comparison of controller performance is shown in Fig. 5, where the relative closed-loop cost and the relative number of steps to reach the setpoint for HJB-RL and NN controllers are evaluated across 50 stratified regions of initial states, each referenced to the LH-LMPC baseline (0%). Negative values in both metrics indicate performance improvement over the baseline, with lower cost corresponding to better control efficiency and fewer control moves indicating reduced actuation effort. As shown in Fig. 5, the HJB-RL controller achieves lower closed-loop costs than the LH-LMPC baseline across all regions of the initial state-space, with the average relative cost remaining negative for each of the 50 stratified regions. This indicates that the HJB-RL policy not only stabilizes the system effectively but also does so with improved control efficiency. Moreover, the HJB-RL controller generally requires fewer control moves than the baseline, reflecting reduced actuation effort. In contrast, the NN controller exhibits higher relative costs and a larger number of control moves across most regions, suggesting that it is less efficient in both control performance and actuation usage. These results demonstrate that the HJB-RL framework can achieve better overall closed-loop performance while maintaining fast control action decisions.

Remark 19. Proportional, FNN, and RL controllers have a nonzero per-step computation time because each must execute specific computations to produce the control input. A proportional controller still has to read the measurement, compute the error, and apply the gain, so its execution time is not strictly zero even though it is very small. A neural network controller must perform a forward pass, which consists of matrix–vector multiplications and activation function evaluations, and this introduces additional computation time that scales with the network's size. Similarly, an RL controller also performs a forward pass to generate the control action and may include extra steps such as critic evaluation or safety projection, which further contribute to the overall computation time.

Remark 20. The computation times shown in Fig. 4 are not GPU-accelerated or parallelized. This is mostly negligible for the LMPC and P-based controllers due to the small system scale, but the lack of tensor-accelerated optimizations is a significant handicap to the FNN and RL-based methods. Thus, the computation times shown are conservative estimates, and real-world use cases may see significantly better improvements with the proper hardware and software optimizations.

4.7.2. TD3-based controller results

To demonstrate the TD3-based controller's quality, we apply this controller to the closed-loop system using 100 randomly generated initial points within the same intervals used in the HJB-based controller's trials. The TD3-based controller is applied for 500 steps, where each step corresponds to a time interval equal to Δ. In other words, the TD3-based controller is used to control the system starting from the initial state until the trajectory spans 500 s worth of control. For each of these trajectories, the cumulative cost is calculated. For each of the initial states, the long-horizon LMPC is simulated to give a reference cost. Using the long-horizon LMPC's cumulative cost, we calculate

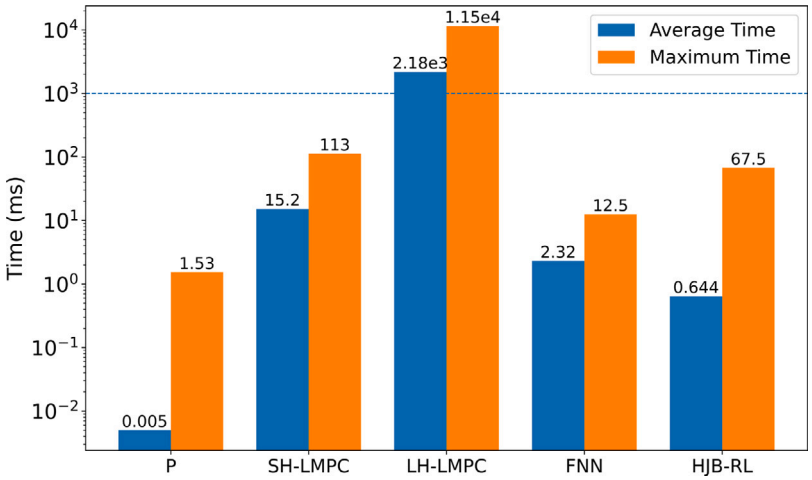
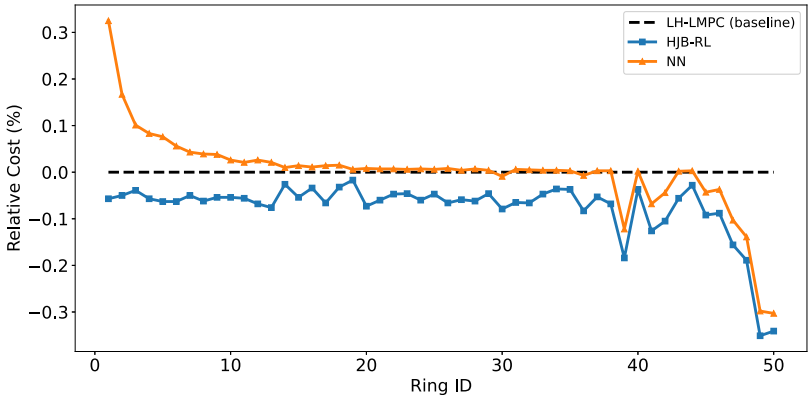
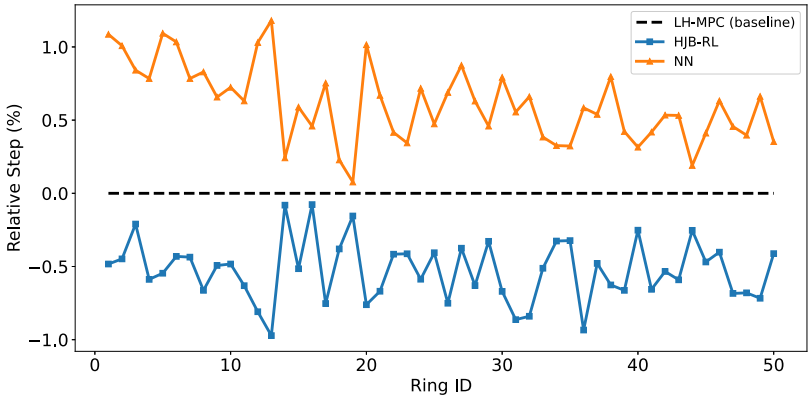


Fig. 4. Per-step computation time of each controller. Blue/orange bars show average and maximum time, respectively, for the P controller, short-horizon LMPC, long-horizon LMPC, FNN, and HJB-RL. The dashed horizontal line denotes the sampling time (1 s = 1000 ms); real-time feasibility requires controller times below this line.



(a) Relative closed-loop cost across 50 stratified rings of initial states for HJB-RL and NN, referenced to the LH-LMPC baseline (0%). Negative values indicate improvement over the baseline.



(b) Relative number of control moves across the same 50 rings for HJB-RL and NN, referenced to the LH-LMPC baseline (0%). Lower values indicate fewer moves than the baseline.

Fig. 5. Controller performance relative to LH-LMPC across stratified initial conditions: (a) cost and (b) control-step counts for HJB-RL and NN.

the relative percent error of the TD3-based controller with respect to this cost. This percent error is averaged over the 100 samples in this evaluation run, which yielded an average percent error of $-0.91\,\% \pm 9.87\,\%$. At first glance, this result appears good, but the standard deviation is quite large. Upon analysis of the individual percent error values, it was found that this high standard deviation is due to poor performance of the model for initial states that are already near the origin. The source of this issue is that the TD3-based controller's control action converges to an offset when near the origin, as can be seen in Fig. 6. Additionally, the controller seemingly struggles to behave in a smooth fashion for more complex control trajectories as seen in Fig. 7, which contributes to this variability. Beyond these outliers,

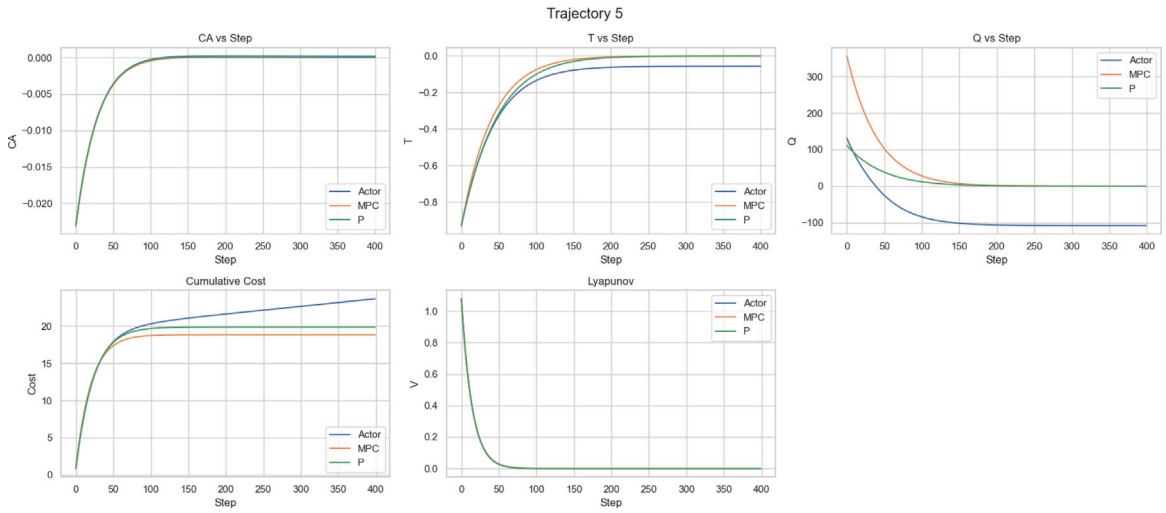


Fig. 6. Closed-loop trajectory for sample #5 of the 100 trajectories used in the calculation of the average MSE for the TD3-based RL-controller. This sample demonstrates the near-origin offset of the TD3-based RL-controller in the absence of fallback control. Steps refer to time intervals of size Δ. C_A, T, and $\dot{Q}$ are in deviation variable form with respect to the steady-states defined in Table 1 and have units of $\mathrm{kmol\,m^{-3}}$, K, and $\mathrm{kJ\,h^{-1}}$ respectively.

the results are seemingly significantly better than the HJB-based RL-controller, but this is misleading. Recall that this closed-loop evaluation run used a TD3-based RL-controller that used the best performing hyperparameters from the Optuna trial shown in Table 5. Further, the training cycle was modified to allow for extended training with the goal of further fine-tuning the model. In reality, the Optuna trial had an average percent error of $-4.5\,\%$, meaning the full-training loop seemingly decreased the model's quality. Thus, the hyperparameters for the full training run were reverted to the Optuna trial hyperparameters, and the same parameters from Table 5 were used for a training run that aimed to replicate what the Optuna trial found. Instead, this yielded a 100 closed-loop average percent error of $2.07\,\% \pm 8.12\,\%$ with a variance of $66.01\,\%$. Not only did the value change, but it went from significantly better than the long-horizon controller to worse while still retaining the flawed fitting demonstrated in Figs. 6 and 7. Since closed-loop error predominantly comes from trials near the origin, the IQR method is applied to filter out outliers, which yielded an 84 closed-loop average percent error of $0.41\,\% \pm 0.4\,\%$ with a variance of $0.17\,\%$. Even ignoring the outliers, the resulting RL-controller not only struggles to maintain consistent performance relative to the long-horizon LMPC, but also fails to perform better for most trials.

Remark 21. Figs. 6 and 7 include plots of the long-horizon LMPC and the fallback Proportional controller as a point of reference. These figures demonstrate desirable properties (smooth transitions between control actions and no offset near the origin) that are satisfied by the controllers that we aim to mimic with the RL-based control design. The HJB-based controller, FNN-based controller, and short-horizon LMPC are not included as they are unnecessary to demonstrate this point and would only clutter the figures with excess information.

Remark 22. Figs. 6 and 7 do not demonstrate real-time implementation of the TD3-based controller to the process due to the lack of the constraint enforcer. Behavior such as the offset demonstrate how, in the absence of the constraint enforcer, closed-loop stability is not guaranteed. Real-time implementation of the TD3-based controller would require the constraint enforcer to correct this behavior.

4.7.3. TD3-based controller hyperparameter tuning analysis

The training of this RL-controller requires that both the Actor and Critic converge before being considered complete; however, the practical implementation of the resulting controller depends on the quality of the Actor. The Critic only serves as a means to assist in the training of the Actor, but in practice, the method struggles to train an Actor so that it can handle higher levels of complexity in the closed-loop trajectories as shown in Fig. 7. One can see that, despite the state trajectories only deviating slightly, the control action trajectory lacks the smoothness found in either the reference Proportional controller or the long-horizon LMPC. The result of this is a cumulative cost over the trajectory that exceeds the long-horizon LMPC. A consequence of this variance is that not all sampled initial states will explore regions in the state-space that require intricate control action patterns, nor will they all be near the origin. Thus, even with 100 samples, the evaluation portion of the training loop would fail to properly quantify the model's performance, leading to training results that are overly optimistic. TD3's reliance on several forms of inherent randomness further distorts the consistency of the results, although this is not as a significant contributing factor compared to the high step count and the use of soft updating steps which dampen the influence of rapid changes. This result is the primary reason why the following analysis will now focus on if the Optuna study can provide insights into how to overcome this variability, instead of a robust closed-loop performance assessment as was done for the HJB-based controller; the TD3-based method is too noisy for consistent results and too difficult to assess the quality of during training.

Optuna provides tools to analyze the parameter importance and the overall sensitivity of the method to hyperparameter optimality. To start, recall that the objective function of this study is the average percent error of the RL-controllers' trajectory cost relative to the long-horizon LMPC baseline over all evaluated trials. Although this term would give a good estimate for performance for a given initial point, it varies significantly depending on the complexity of the dynamics encountered along the closed-loop trajectory for any given starting point. Additionally, the non-zero risk of the LMPC system failing to solve, any potential inaccuracies in our reference controller design, and any regional inaccuracies of the RL-controller can result in a percent error value that can also vary significantly between evaluations. To attempt to mitigate this issue for any given trial, the mean percent error is averaged across the 3 most recent evaluation runs. A key exception to this is that trials that fully complete the training run without being pruned will report the percent error value from the final set of closed-loop runs only instead of using a moving average approach. This was done with the intent of isolating good-quality runs, but had the unintended consequence of reintroducing the variability issue as well as including trials whose convergence was slow but progressive enough to not trigger any early stopping or pruning. As

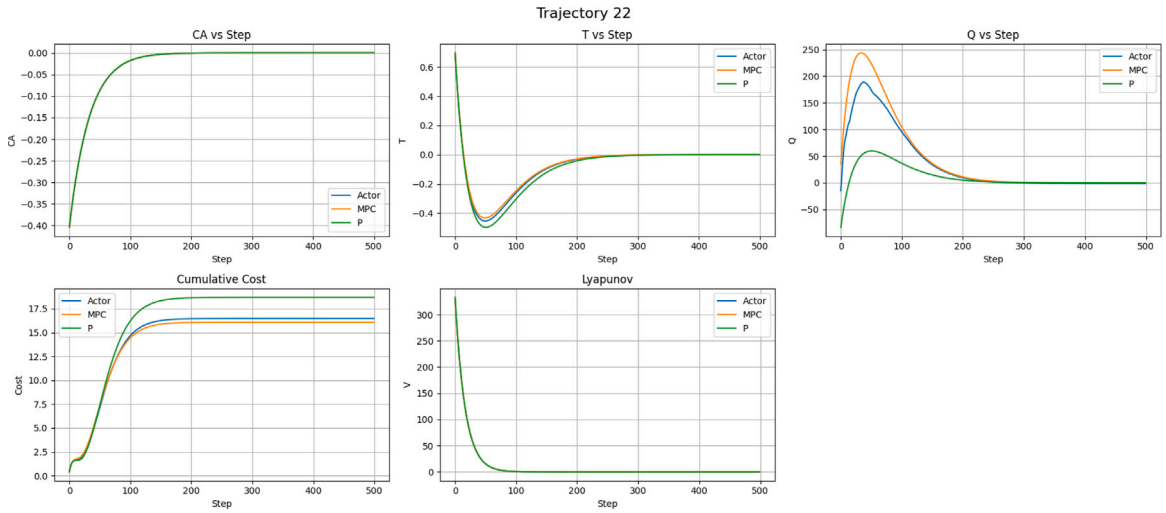


Fig. 7. Closed-loop trajectory for sample #22 of the 100 trajectories used in the calculation of the average MSE for the TD3-based RL-controller. This sample demonstrates the coarseness of the TD3-based RL-controller in the absence of fallback control for regions of the state-space where the dynamics are more complicated. Steps refer to time intervals of size Δ. C_A, T, and $\dot{Q}$ are in deviation variable form with respect to the steady-states defined in Table 1 and have units of $\mathrm{kmol\,m^{-3}}$, K, and $\mathrm{kJ\,h^{-1}}$ respectively.

Fig. 8. Evolution of the best objective values under Optuna trials.

a result, the Optuna trial demonstrated a steady rate of improvement over time for the best final objective value, as shown in Fig. 8. It is important to point out that while TD3 aimed to reduce the issue of exploitation, this steady improvement indicates that the Optuna study may have exploited the variability of the evaluation loop itself. To determine if the study truly failed due to exploitation of the evaluation noise, we must observe the hyperparameter interconnectivity and compare the results to what is known in RL theory. Most importantly, the behaviors should demonstrate that the bias/variance balance and exploration/exploitation balance are being considered.

Optuna provides several tools for further analysis of the results as a means to aid in any extra fine-tuning that may be done. A key tool is the "importance" of the tuned hyperparameters. Like the LMPC, the hyperparameter tuning process suffers from the curse of dimensionality, and so it is a good practice to narrow the number of parameters to tune and the range of these parameters. Unfortunately, this is difficult for the TD3 method, as there are a large number of hyperparameters that need to be considered. With a robust enough study, one can use the importance metrics to determine for future cases which parameters are worth optimizing. Table 4 demonstrates how the Optuna study that was carried out for the TD3-based system is sufficiently broad to encompass the key hyperparameters for the TD3 method. By modifying the study, it becomes possible to extract Optuna's native importance analysis methodology and apply it to the top quartile of trials in the study independently. Fig. 9 demonstrates the resulting importance values of the hyperparameters for the global scope of the study and for the top quartile of the study. This is beneficial as it allows for an understanding of parameters that should be tuned initially for broader performance (overall, global scope) as well as parameters that should be further fine-tuned to truly maximize the performance of the method (top quartile scope).

The results of this chart demonstrate the importance, or lack thereof, of the modifications that TD3 introduces over the standard DDPG algorithm at the global scale. Of the three modifications, there is an

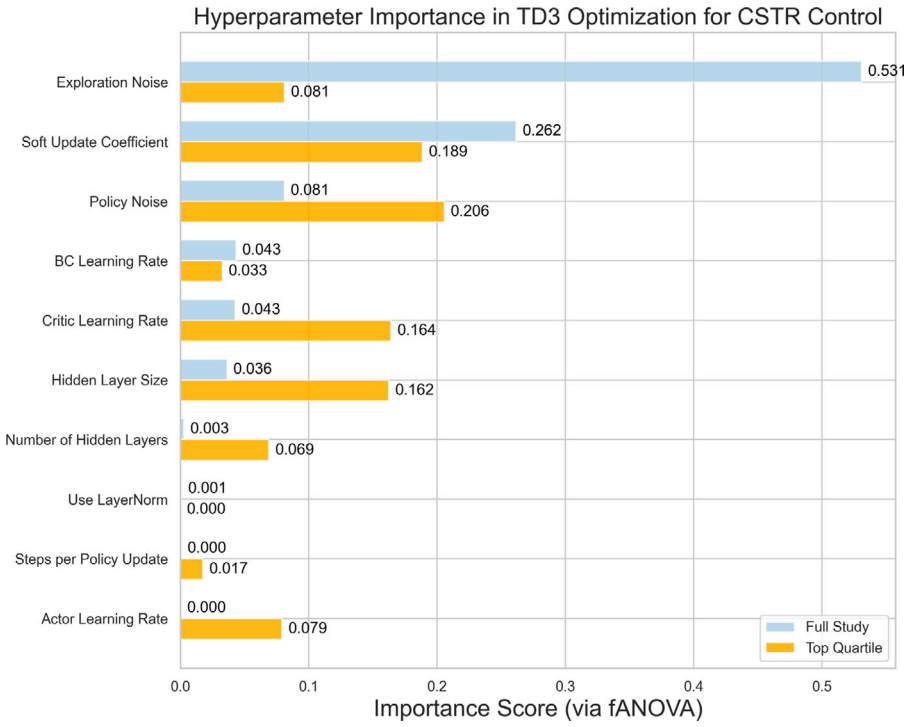


Fig. 9. Organized chart of the hyperparameter importance values. Optuna uses a random forest regression model to estimate the objective based on hyperparameter values. A fANOVA algorithm is then used to derive an 'importance' metric. This metric roughly describes how the hyperparameter influences the variance of the objective, thereby making it 'important' to tune.

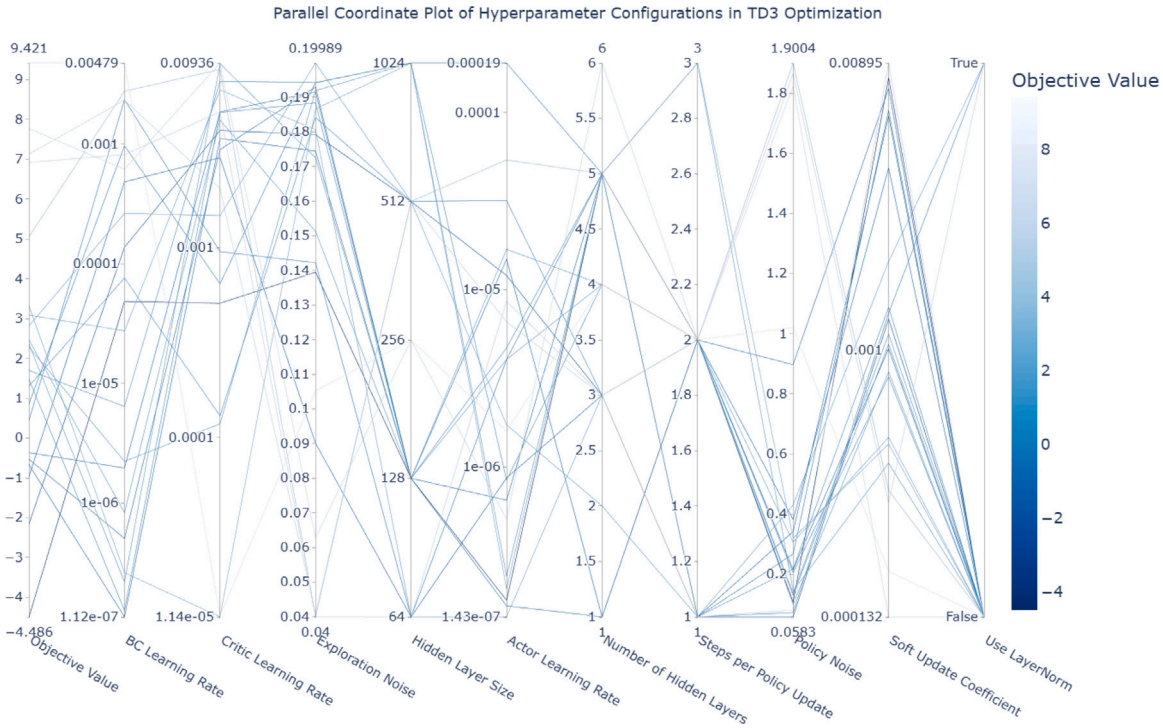


Fig. 10. Parallel coordinate plot of the hyperparameters across all trials. Darker blue lines signify that the objective value for the corresponding hyperparameters was of higher quality.

implied significance from the dual-Q design via the importance of the Critic learning rate, and there is notable significance from the policy noise term. Most surprisingly, the policy update delay is found to have negligible influence. Thus, on a global scale, the TD3 modifications do not necessarily dictate the model's performance. Instead, this demonstrates that the baseline performance is heavily influenced by the degree to which exploration is allowed, and the corresponding degree to which the results are exploited by the Critic.

On the top quartile scale, these results change. Instead of a focus on the exploration vs exploitation dilemma as is common for RL tasks,

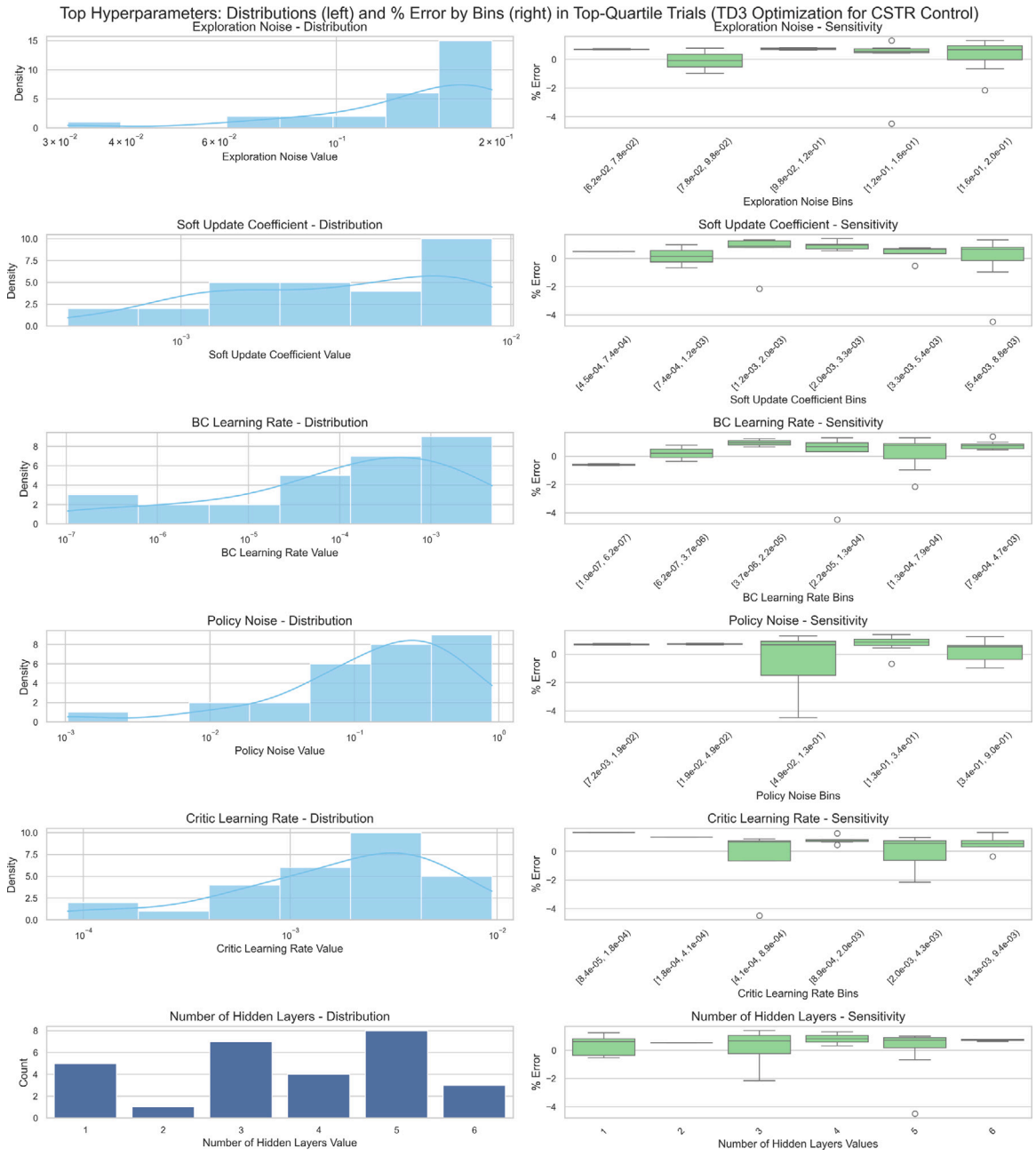


Fig. 11. Histogram (Left) and box plot (Right) of hyperparameters from the top-quartile of trials sorted by their global importance from Fig. 9.

the factors that influence the fine-tuned performance of the model emphasize regularization, model complexity, and the learning rates. This too is an understandable trend, as the balance between these three behaviors is what enables finer fitting to more complex behaviors; however, the importance of these terms tells an incomplete story. To better understand the meaning of these results, the distribution of these hyperparameters needs to be further analyzed. Figs. 10–12 aim to visualize this information.

Fig. 10 demonstrates that a moderate to high Critic learning rate, higher exploration noise, and large soft update coefficients lead to better objective values. Fig. 11 validates these findings. Due to the use of the OneCycleLR scheduler, all learning rates represent the maximum learning rate, where the true learning rate begins at 1/25 of this value before rapidly reaching the max and gradually decaying to 1×10^{-4} times the max. Thus, it is understandable that moderate to high learning rates are preferred over lower values, as this enables the use of a broader range of learning rates. The model will aggressively search in the beginning, followed by incrementally slowing the learning process near the end of the trial to allow for refinement. Similarly, the exploration noise term is designed to decay as the process proceeds. The rationalization of the soft-update coefficient behavior is less clear. It is possible that less dampening via the target models better complements the slow exploration and learning of later steps, or perhaps the need for such aggressive dampening is a hindrance to the model's learning process after proper pretraining. The figure further demonstrates that moderate to high policy noise and Critic learning rates improve the objective values. While the Critic's learning rate behavior matches the explanation regarding the scheduler above, the policy noise does not decay like the exploration noise. Such high policy noise might explain the trend with the soft update coefficient; in order to compensate for the loss of dampening via soft weight updates, the policy noise, the other mode of countering exploitation in the TD3 algorithm, becomes higher. Finally, the Actor pre-training learning rates lack an obvious trend. This term is important for ensuring good pretraining, but this

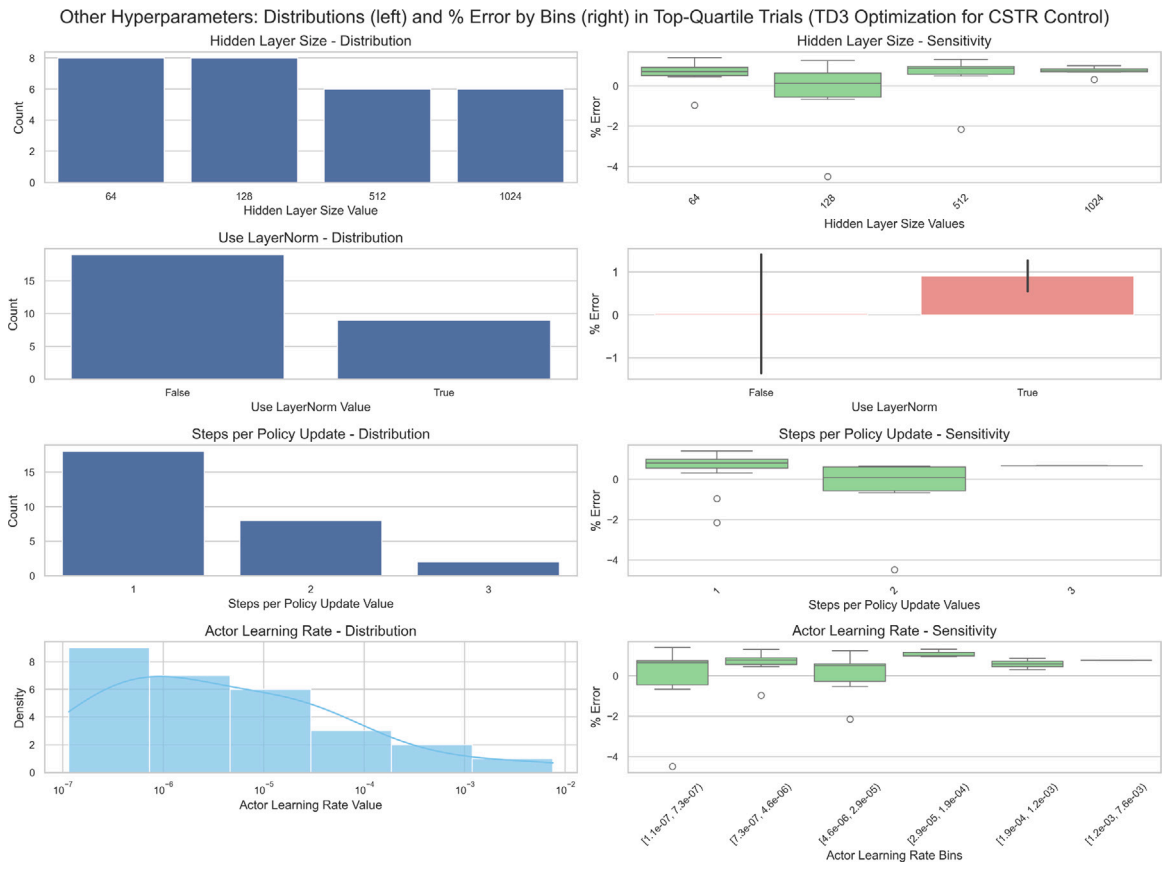


Fig. 12. Histogram (Left) and box plot (Right) of the remaining hyperparameters from the top-quartile of trials, sorted by their global importance from Fig. 9.

can be achieved with a separate Optuna study, so further analysis of this term is not of meaningful value.

Fig. 12 contains all the parameters that are of low importance globally. The key takeaway from this plot is that LayerNorm worsens the training, and that the single-step delay and lower Actor learning rates are helpful. Of these, the Actor learning rate is the most important for the top quartile of runs, and the trend of favoring low values leads to a smoother training process for the Critic, as higher learning rates would compound with the policy noise. Fig. 13 gives insights into how these terms appear to influence one another as well as how they influence the objective value. Of the various takeaways from this figure, the high correlation between the hidden layer size and the bulk of the hyperparameters is an interesting result. In the top quartile of runs, higher complexity in the networks was correlated with both higher exploitation via high learning rates and less dampening, as done in TD3/DDPG, and higher regularization via LayerNorm. It is also correlated to worse results, which aligns with the prior plots that show model complexity is important up to a certain point, at which point further complexity hinders the algorithm's performance. These results further demonstrate that higher policy noise, which hinders exploitation, is matched with less Actor learning, but more exploration and more Critic learning. The higher risk of exploitation from the low policy noise is compensated by simultaneously increasing exploration and lowering regularization for the critic. In a sense, this demonstrates that for the provided set-up, the benefits of the TD3 algorithm are minimal at the global scale but critical for getting the best objective. Only the twin-Q aspect remains inconclusive, as this was not tested.

While the results may roughly agree with RL theory, it is important to note that extracting trends from the limited samples of this study is inconclusive with respect to the true trends; however, the current evidence does not support the assertion that the Optuna study suffered from exploitation of the noisy evaluations. Thus, the poor behavior

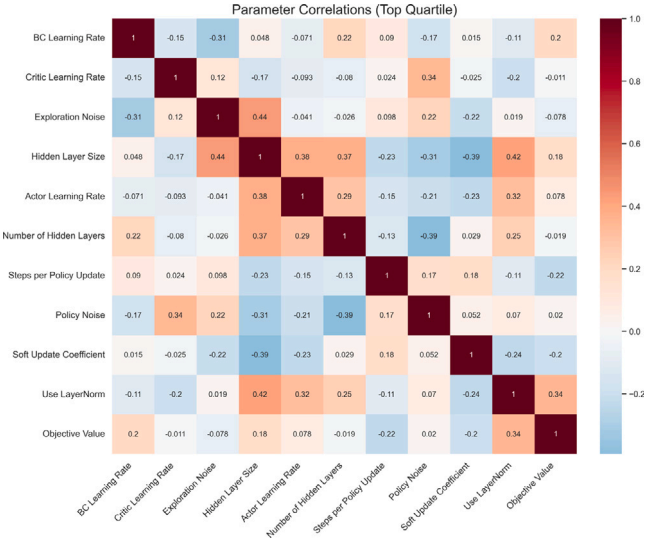


Fig. 13. Heatmap of hyperparameters and objective value from the top quartile of trials. Large magnitude values indicate high correlations between the terms on the respective row and column.

stems from either an insufficient number of trials in the Optuna study or from problems stemming from the TD3 training loop itself. Note that the training loop uses early stopping via closed-loop evaluation, thereby preventing full convergence of the temporal difference loss term as well as the actor's loss, which prevents true convergence of the system. The lack of full convergence is of particular concern with regard to the actors in these trials, which tended to have low learning

rates on top of the existing scheduler decay and soft update dampening. Thus, despite the tuning process roughly following the trends of RL theory, the scope of hyperparameters that have high significance and need to be finely tuned, paired with the lengthy convergence process of the TD3 algorithm, leads this to be an ill-suited approach for those looking for an easy conversion to RL-based process control. There is, however, promise in using this methodology to derive an RL-based controller that exceeds the performance of long horizon LMPC, as some trials yielded a better average than the HJB-based controller, which trends towards the LMPC's behavior. Thus, even if the near-origin inaccuracies of the TD3-based controller are not solved with finer tuning, the controller can potentially serve as the controller for values far from the origin, with the HJB-based controller handling the values near the origin. As a model-free off-policy method, TD3 has the potential to find performance gains, but reaching this point consistently may be a challenge.

5. Conclusion

In this work, we proposed a stabilizing reinforcement learning (RL)-based control framework. Specifically, we enforce stability online via a per-step Lyapunov decrease test that compares between control actions from the RL-based controller ($\Phi_{\mathrm{RL}}(x)$) and a reference stabilizing controller. When $\Phi_{\mathrm{RL}}(x)$ is less contractive (with respect to the decay rate of a Lyapunov function for the closed-loop system) than the reference controller for a given state, a short-horizon model predictive controller with a contractive Lyapunov constraint (LMPC) is applied. This framework is applied to two forms of RL, one using a modified TD3 algorithm and the other using an HJB-based method. When the HJB-based RL-controller is applied to a simulated continuous stirred tank reactor case study, the proposed RL-based controller demonstrates stable setpoint tracking with superior performance. In comparison with long-horizon LMPC, short-horizon LMPC, a reference stabilizing Proportional controller, and an FNN-based approximate LMPC, the RL-based controller achieved better setpoint-tracking performance than the long-horizon LMPC while maintaining fast computation comparable to FNN-based approximate LMPC, thereby enabling real-time control with potentially improved control quality and reduced computational burden. In contrast, the TD3-based RL-controller demonstrated that the method could yield notably better results than the long horizon LMPC with the exception of poor fitting to points near the origin; however, the tuning of such a controller is not only more important for the final closed-loop performance, but it is also more difficult to tune effectively due to the high sensitivity and interdependencies of the method's extensive list of hyperparameters.

CRediT authorship contribution statement

Arthur Khodaverdian: Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Xiaodong Cui:** Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Panagiotis D. Christofides:** Writing – review & editing, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

Financial support from the National Science Foundation, United States, CBET-2140506, is gratefully acknowledged.

References

Adabag, E., Atal, M., Gerard, W., Plancher, B., 2024. MPCGPU: Real-time nonlinear model predictive control through preconditioned conjugate gradient on the GPU. In: Proceedings of International Conference on Robotics and Automation. Yokohama, Japan, pp. 9787–9794.

Ahn, K., Mhammedi, Z., Mania, H., Hong, Z.W., Jadbabaie, A., 2022. Model predictive control via on-policy imitation learning. arXiv preprint arXiv:2210.09206.

Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M., 2019. Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.

Alora, J.I., Pabon, L.A., Köhler, J., Cenedese, M., Schmerling, E., Zeilinger, M.N., Haller, G., Pavone, M., 2023. Robust nonlinear reduced-order model predictive control. In: Proceedings of 62nd Conference on Decision and Control. Marina Bay Sands, Singapore, pp. 4798–4805.

Alsmeier, H., Savchenko, A., Findeisen, R., 2024. Neural horizon model predictive control - increasing computational efficiency with neural networks. In: Proceedings of the American Control Conference. Toronto, Canada, pp. 1646–1651.

Bonzanini, A.D., Paulson, J.A., Graves, D.B., Mesbah, A., 2020. Toward safe dose delivery in plasma medicine using projected neural network-based fast approximate NMPC. IFAC-PapersOnLine 53, 5279–5285.

Fiedler, F., Karg, B., L"uken, L., Brandner, D., Heinlein, M., Brabender, F., Lucia, S., 2023. Do-mpc: Towards fair nonlinear and robust model predictive control. Control Eng. Pract. 140, 105676.

Gill, P.E., Murray, W., Picken, S.M., Wright, M.H., 1979. The design and structure of a fortran program library for optimization. ACM Trans. Math. Soft. 5, 259–283.

Gonzalez, C., Asadi, H., Kooijman, L., Lim, C.P., 2024. Neural networks for fast optimisation in model predictive control: A review. arXiv preprint arXiv:2309.02668.

Gordon, D.C., Winkler, A., Bedei, J., Schaber, P., Pischinger, S., Andert, J., Koch, C.R., 2024. Introducing a deep neural network-based model predictive control framework for rapid controller implementation. In: Proceedings of American Control Conference. Toronto, Canada, pp. 5232–5237.

Khodaverdian, A., Gohil, D., Christofides, P.D., 2025a. Enhancing cybersecurity of nonlinear processes via a two-layer control architecture. Digit. Chem. Eng. 15, 100233.

Khodaverdian, A., Gohil, D., Christofides, P.D., 2025b. Neural network implementation of model predictive control with stability guarantees. Digit. Chem. Eng. 16, 100262.

Khodaverdian, A., Gohil, D., Christofides, P.D., 2026. Uniting neural network-based control and model predictive control: Application to a large-scale nonlinear process. Comput. Chem. Eng. 204, 109396.

Kraft, D., 1988. A software package for sequential quadratic programming. In: Deutsche Forschungs- Und Versuchsanstalt FÜr Luft- Und Raumfahrt KÖln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR.

Lewis, F.L., Vrabie, D., Syrmos, V.L., 2012. Optimal Control. John Wiley & Sons, Hoboken, NJ, USA.

Lucia, S., Karg, B., 2018. A deep learning-based approach to robust nonlinear model predictive control. IFAC-PapersOnLine 51, 511–516.

Macmurray, J., Himmelblau, D., 1995. Modeling and control of a packed distillation column using artificial neural networks. Comput. Chem. Eng. 19, 1077–1088.

Meng, D., Chu, H., Tian, M., Gao, B., Chen, H., 2024. Real-time high-precision nonlinear tracking control of autonomous vehicles using fast iterative model predictive control. IEEE Trans. Intell. Veh. 9, 3644–3657.

Mhaskar, P., El-Farra, N.H., Christofides, P.D., 2006. Stabilization of nonlinear systems with state and control constraints using Lyapunov-based predictive control. Syst. Contr. Lett. 55, 650–659.

Pardalos, P.M., Vavasis, S.A., 1991. Quadratic programming with one negative eigenvalue is np-hard. J. Global Optim. 1, 15–22.

Patel, R., Bhartiya, S., Gudi, R.D., 2025. Neural network-based model predictive control framework incorporating first-principles knowledge for process systems. Ind. Eng. Chem. Res. 64, 9287–9302.

Peng, Y., Yan, H., Rao, K., Yang, P., Lv, Y., 2024. Distributed model predictive control for unmanned aerial vehicles and vehicle platoon systems: a review. Intell. Robot. 4, 293–317.

Qin, S.J., Badgwell, T.A., 2003. A survey of industrial model predictive control technology. Control Eng. Pract. 11, 733–764.

Ren, Y.M., Alhajeri, M.S., Luo, J., Chen, S., Abdullah, F., Wu, Z., Christofides, P.D., 2022. A tutorial review of neural network modeling approaches for model predictive control. Comput. Chem. Eng. 165, 107956.

Tomasetto, M., Braghin, F., Manzoni, A., 2025. Latent feedback control of distributed systems in multiple scenarios through deep learning-based reduced order models. Comput. Methods Appl. Mech. Engrg. 442, 118030.

Wu, Z., Tran, A., Rincon, D., Christofides, P.D., 2019. Machine learning-based predictive control of nonlinear processes. part I: Theory. AIChE J. 65, e16729.

Yaren, T., Kizir, S., 2025. Real-time nonlinear model predictive control of a robotic arm using spatial operator algebra theory. J. Field Robot. (in press).

Zarrouki, B., Nunes, J., Betz, J., 2023. R^2NMPC: A real-time reduced robustified nonlinear model predictive control with ellipsoidal uncertainty sets for autonomous vehicle motion control. arXiv preprint arXiv:2311.06420.