SAC for DMC coordination

Anika Patel, Arian Patel, Kenneth Ma, Natael Predestin Jr.

Background

Dynamic Matrix Control (DMC) is widely used to regulate individual processes in chemical plants. However, coordinating multiple DMC controllers remains a challenge due to nonlinear interactions that can cause instability and inefficiencies (Georgiou, et al, 1988). This project aims to develop a reinforcement learning (RL) agent capable of managing multiple DMC controllers in a simulated offline environment. The agent will learn optimal coordination strategies that enhance process stability, energy efficiency, and plant performance.

DMC is a form of Model Predictive Control (MPC) that maximizes process objectives subject to constraints. It manipulates system inputs while predicting future conditions to meet control objectives. DMC uses a finite impulse response (FIR) model to predict how changes in manipulated variables will affect control variables.

For a single-input, single-output system, this is (Georgiou, et al, 1988):

$$y(k) = \sum_{j=1}^{N} g_j \Delta u(k-j) + y_0(k)$$

k, j: time
y: output prediction function
g: step response coefficient
u: input reading
y_o: free response function

The control objective is described by a cost quadratic function that is minimized over a prediction horizon P (Garcia, et al, 1982):

$$J = \sum_{i=1}^{P} \left[y(k+i|k) - y_{\text{set}}(k+i) \right]^2 + \lambda \sum_{i=0}^{M-1} \left[\Delta u(k+i) \right]^2 .$$

i, P, M: time

DMCs can be extended to multivariate systems. Optimizing groups of these systems are the focus of this project.

The Generic Reinforcement Learning Library (GRL) has successfully simulated chemical plant environments with realistic plant dynamics, system constraints, and process variability.

Deep Q-Learning (DQL) has demonstrated strong performance in similar process control settings. Luz and Caarls showed that DQL could achieve results comparable to Nonlinear Model Predictive Control (NMPC) when applied to a Continuous Stirred Tank Reactor (CSTR), with the added benefit of significantly reduced computational costs after training. Their DQL implementation effectively handled setpoint transitions, minimized oscillations, and maintained stability even in challenging regions near gain inversion.

SAC's entropy-based approach balances exploration and exploitation, making it particularly suitable for continuous control problems with complex, nonlinear dynamics, such as those found in coordinated multi-DMC (Dynamic Matrix Control) settings. However, in Luz and Caarls's implementation, SAC underperformed relative to DQL, highlighting a gap between theoretical promise and practical outcomes. This work aims to bridge that gap by revisiting SAC's implementation and investigating its potential to outperform DQL in demanding process control scenarios.

The reinforcement learning agent "explores" the environment by setting different goals for the DMCs. The agent cannot fine-tune the inner workings of the DMCs themselves - these DMCs internally have non-linear processes (like PID) to reach the goals from the current state.

Data generation

The agent has access to the temperature, pressure, K_{eq}, and the constraints of each DMC. It can't manually control the DMC outside of setting the goal temperature; any functions within the DMC are a "black box".

Data generation classes

DMC_structure is an interface to access the data from each DMC, initialized with an array of DMCs. This array is referred to as a factory. DMC_controller is a generic class that defines a given DMC. This class calls iterate(), which updates the data for each DMC in DMC_controller per time step. In a factory setting, DMC_controller would be replaced with data streams for each respective DMC. DMC_structure.iterate() would be replaced with a data request for each DMC.

- DMCs have a maximum temperature change that they can achieve per time step. It was calculated as T_{set}-$T_{current}$, with randomized additions. T_{next} was in the same order of magnitude across all DMCs.

$$change = \begin{cases} T_{set} - T_{current} & : x \leq 0 \\ \frac{T_{set} - T_{current}}{10} & : x > 0 \end{cases}$$

$$T_{next} = T_{current} + change + \varphi$$

- The K_{eq} functions varied slightly across each DMC while maintaining an order of x^{-2}.

$$K_{eq} = \alpha \left(\frac{1}{(\frac{T_{next}}{\beta} - \gamma)^2 + \delta} \right)$$

- The volume, V, was varied across all DMCs with a randomized constant epsilon.

$$V = 5\pi * \epsilon$$

- P_{next} is dependent on the ideal gas constant (R), T_{next} , Volume, and a randomized constant.

$$P_{next} = \frac{(R * T_{next})}{V} + \psi$$

- All factory input streams are static (temperature, pressure, and K_{eq} are constant).

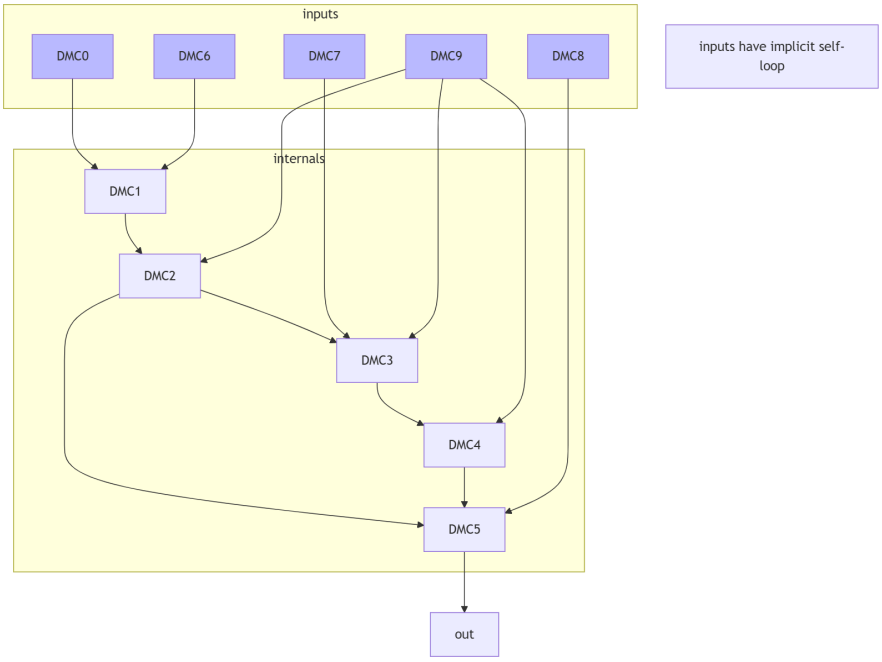


Figure 1: Visualization of a sample factory. All inputs in the model implicitly feed into themselves. The DMC array's data structure is a directed acyclic graph (DAG).

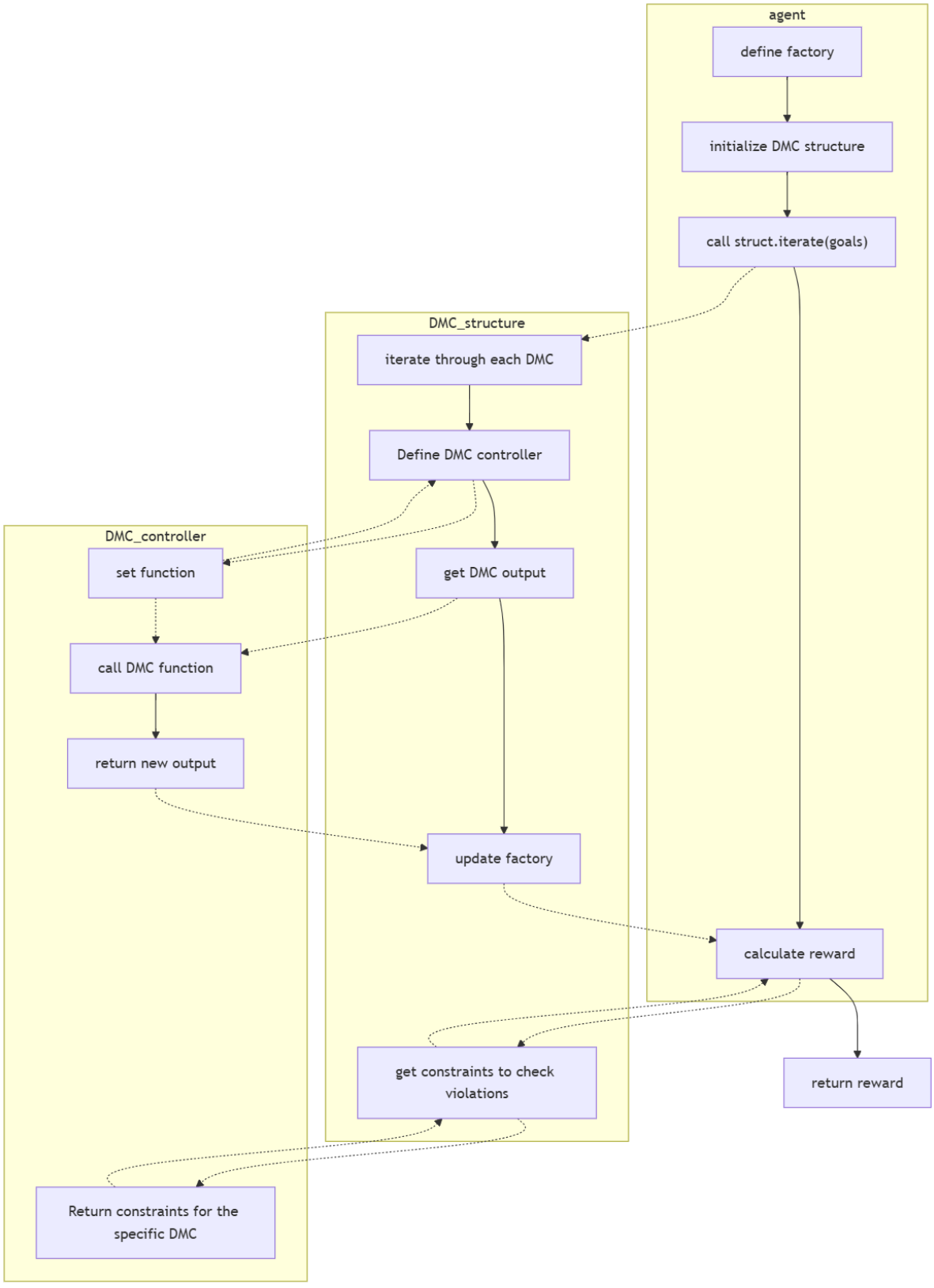


Figure 2: Visualization of the information flow for the DMCs and agent.

Below is a sample run-through for a factory with two DMCs. The input to the first DMC changes to 200 degrees at step 50. This illustrates the following points:

1. Our DMC models sufficiently handle randomness without demonstrating random walk behavior.
2. Our DMC models properly handle large temperature changes.
3. Our DMC models realistically handle not being able to change temperature past a certain point. In other words, it demonstrates the physical limitations of the heating/cooling mechanisms.

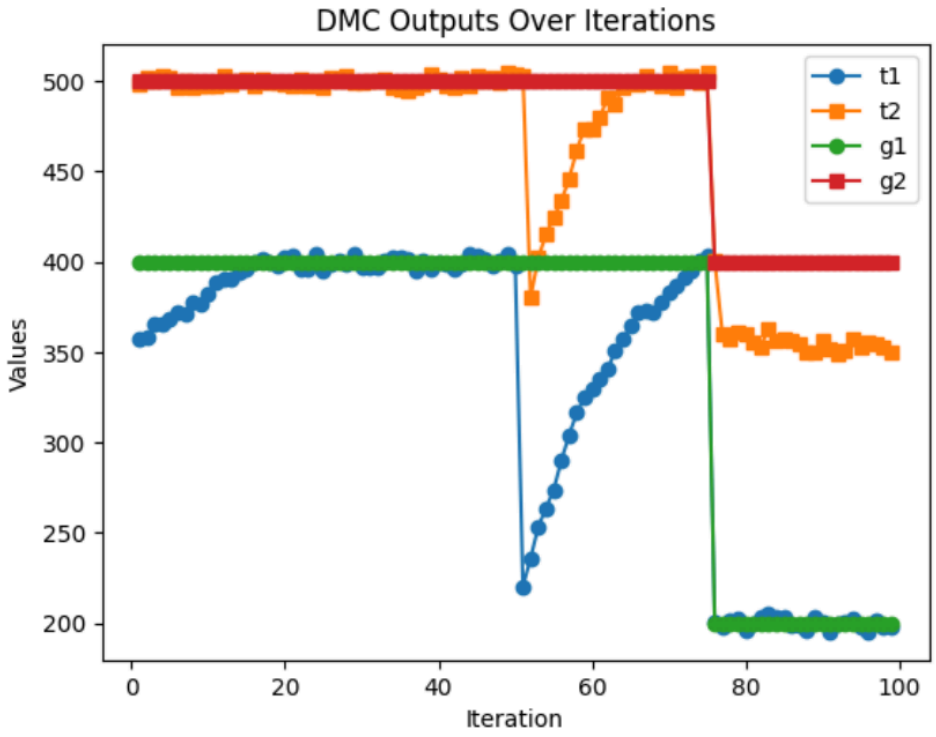


Figure 3: Graph of DMC temperature outputs and goals over 100 iterations. The red and green lines represent the goals, and the orange and blue lines represent the DMC outputs.

Baseline model results

Factory setup

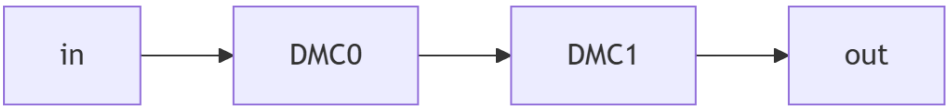


Figure 4: DMC factory for baseline model.

Reward function

The reward function considers a positive reward based on the output of the system and a negative reward based on violating the constraints of the system (i.e., manipulating the DMCs such that the system has a temperature of 500C when it should at most be 475C).

Let the reward function be defined as:

$$R = \underbrace{m \cdot y}_{\text{positive reward}} \underbrace{- \sum_{i=1}^{N} \sum_{j=1}^{M_i} \text{penalty}_{ij}}_{\text{constraint penalties}}$$

$m = 10.0$*: reward scaling factor*
$y = \text{output}[2]$*: output value of interest (*K_{eq}*)*
N*: number of DMC controllers.*
M_i*: number of constraints associated with the* i*-th DMC*

The penalty is defined as follows:

$$\text{penalty}_{ij} = \begin{cases} LB_{ij} + \text{buffer}_{ij} < x_{ij} < LB_{ij} + \text{buffer}_{ij}, & 0 \\ otherwise, & 1 \end{cases}$$

$$\text{buffer}_{ij} = \begin{cases} LB_{ij} = -\infty \vee UB_{ij} = \infty, & 0 \\ otherwise, & 0.1 * (UB_{ij} - LB_{ij}) \end{cases}$$

x_{ij}*: current value*
LB_{ij}*: lower bound*
UB_{ij}*: upper bound*

Hence, the final reward is:

$$R = 10 \times \text{output}[2] - \sum_{i=1}^{N} \sum_{j=1}^{M_i} \text{penalty}_{ij}$$

Baseline Model architecture

The model architecture has two input nodes, followed by two fully connected 16-node layers (ReLU), which combine to a single output with sigmoid activation.

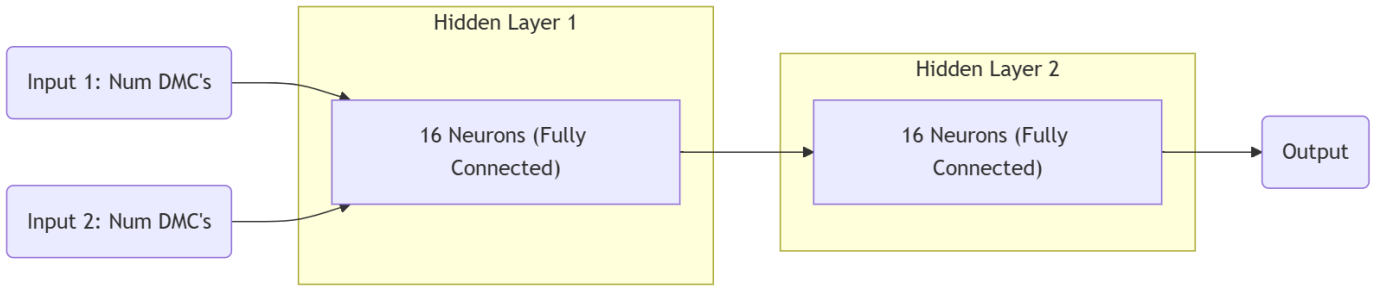


Figure 5: Diagram of model architecture.

Baseline Model results

We did a naive exploration (an approximation of Q-learning) by generating 1000 data points and a regression based on a shallow neural network.

Trial, error, exploration, and then training off of the exploration is typical for reinforcement learning. However, due to static conditions and naive exploration, we trained off of the explored data as a labeled batch, like supervised learning.

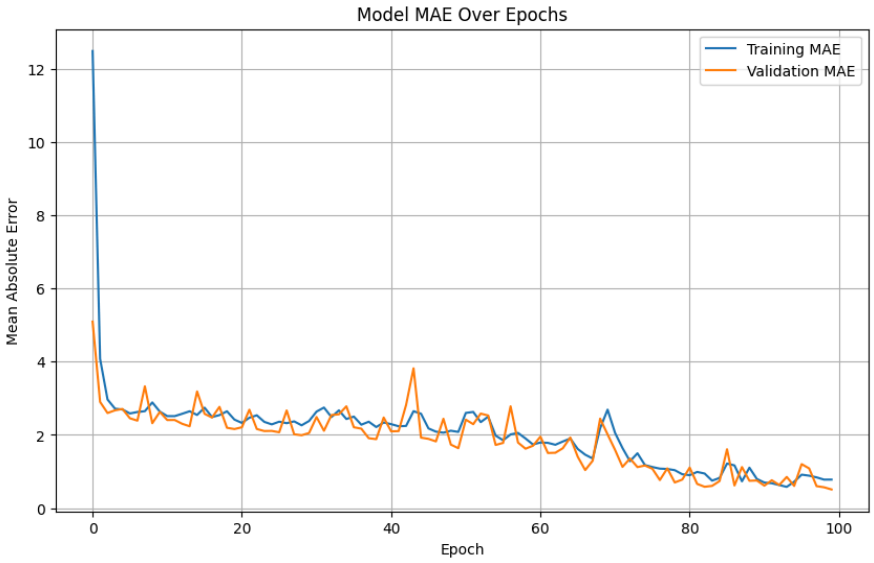


Figure 6: Mean Absolute Error (MAE) over 100 training epochs for both training and validation datasets.

The plot shows that both training and validation MAE decrease steadily over 100 epochs, indicating effective learning and generalization. The close alignment of the two curves suggests minimal overfitting, and the final validation MAE of 0.5096 reflects decent predictive accuracy. Despite minor fluctuations, the model performed consistently well.

Baseline Discussion

We've found that the baseline reward function needs tuning. Optimal predictions like 897/995 and 500/998 are unstable and frequently fall outside the system's bounds of (250, 500) and (400, 600). Different combinations of reward functions didn't get satisfactory results; moving to a continuous reward function may help stabilize the solution.

Training was done using a naive exploration approach, generating data at random points from fixed start conditions, which made this more of a supervised learning problem. The final MSE and MAE were low and stable, and the small gap between training and validation suggests the model generalized well. MSE helped guide the training, while MAE provided an intuitive sense of average error.

Still, due to the naive exploration and simplistic reward function, finding an optimal value turned into a search through the learned model rather than true optimization.

Improved model

For the improved model, we used Soft Actor-Critic (SAC), which is well-suited for this kind of stateful control problem. Solving the problem in a stateless way would reduce it to simply outputting a list of optimal temperatures based on static inputs, which doesn't allow the agent to respond to changing system conditions. By treating it as stateful, we enable the agent to adjust dynamically over time, learning how to make decisions that consider both the current state and the past behavior of the system.

Because the DMC temperature setpoints are continuous variables (within bounds like T_1 to T_2), a discrete-action method like a Q-table isn't appropriate. SAC is a good fit here because it supports continuous action spaces and balances exploitation and exploration using entropy regularization. It consists of an actor network (which proposes actions) and a critic network (which evaluates them), allowing the agent to learn a stable policy over time.

The full training pipeline includes a custom Gym environment that defines the action and observation spaces and calculates rewards, a replay buffer that stores past experiences to decorrelate updates and improve stability, and the SAC model itself, which is trained in a loop. This setup lets the agent explore a wide range of states and actions while learning an efficient policy that can handle variability in system behavior.

Factory setup

The factory for the improved model is much more complex, as seen in the visualization below.

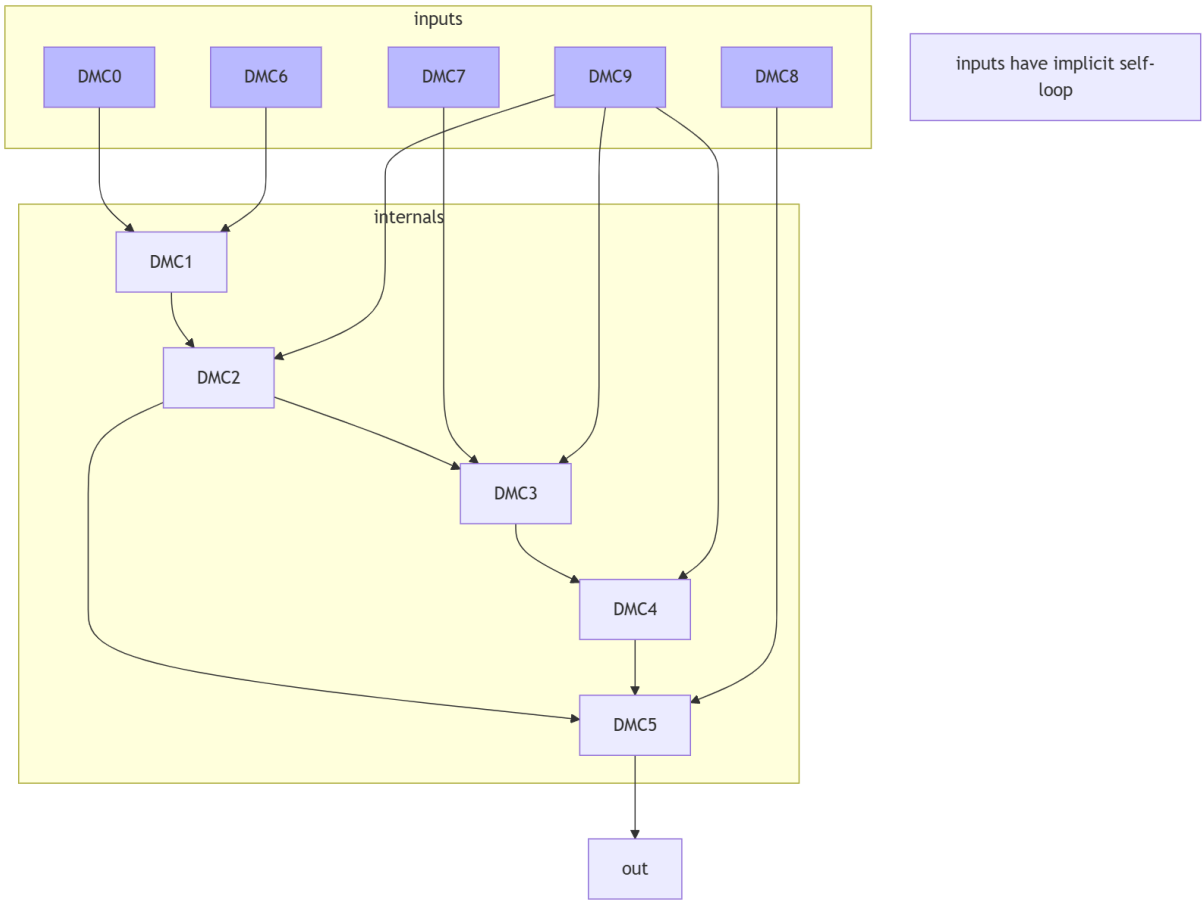


Figure 7: Visualization of DMC web, DMCs “feed” the tail of the arrows

A static simulation of the DMC web in the factory (Figure 8) demonstrates how the various controllers interact with each other.

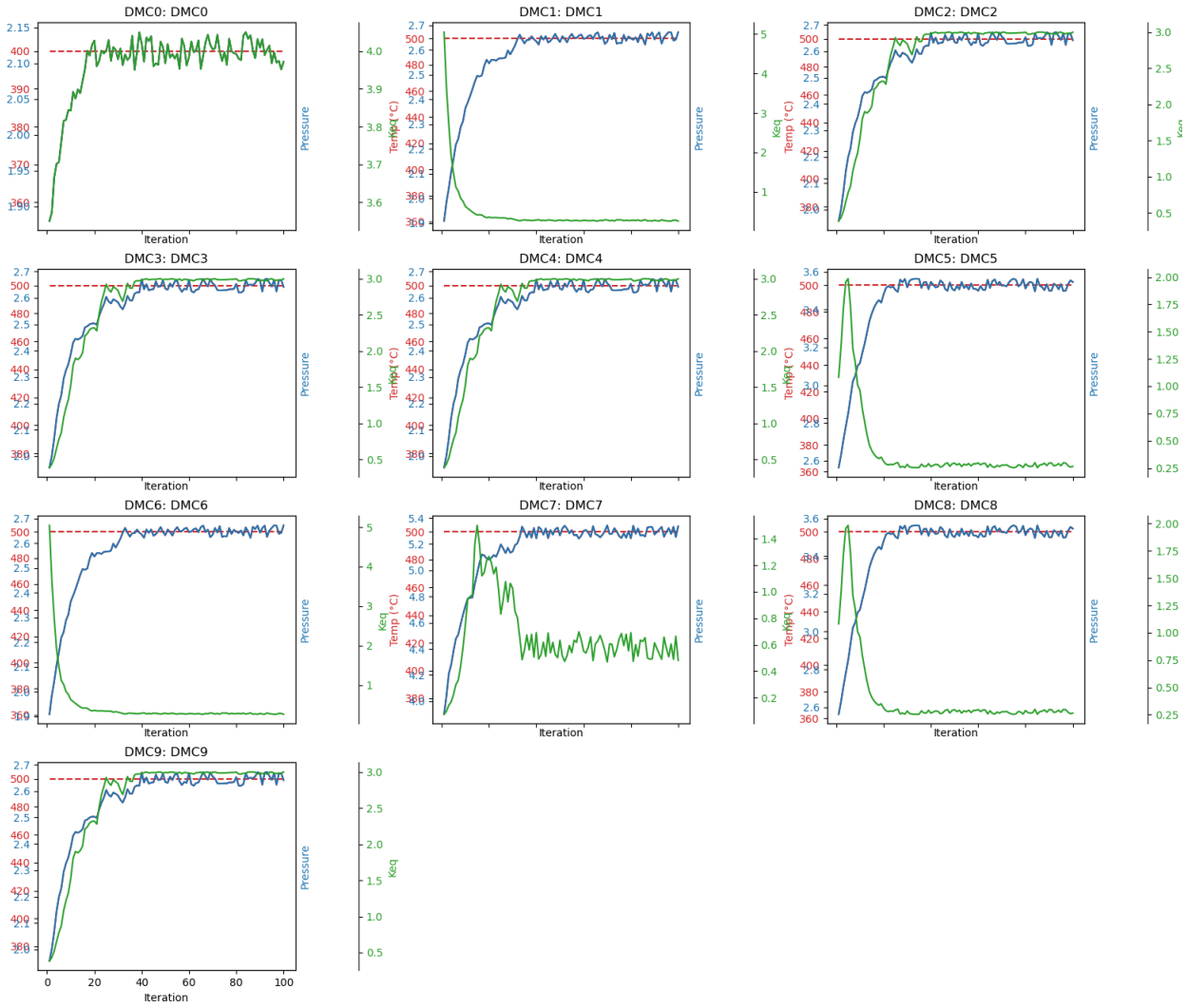


Figure 8: Plot of the temperature (red), pressure (blue), and K_{eq} of all 10 DMCs over 100 steps.

Shown in Figure 8, DMC 1, 5, 7, and 8 best illustrate the nonlinearity added to the K_{eq} outputs. From the web in Figure 7, the sequence of DMCs has complex dependencies. These dependencies can be further manipulated by changing goals, shown in Figure 9.

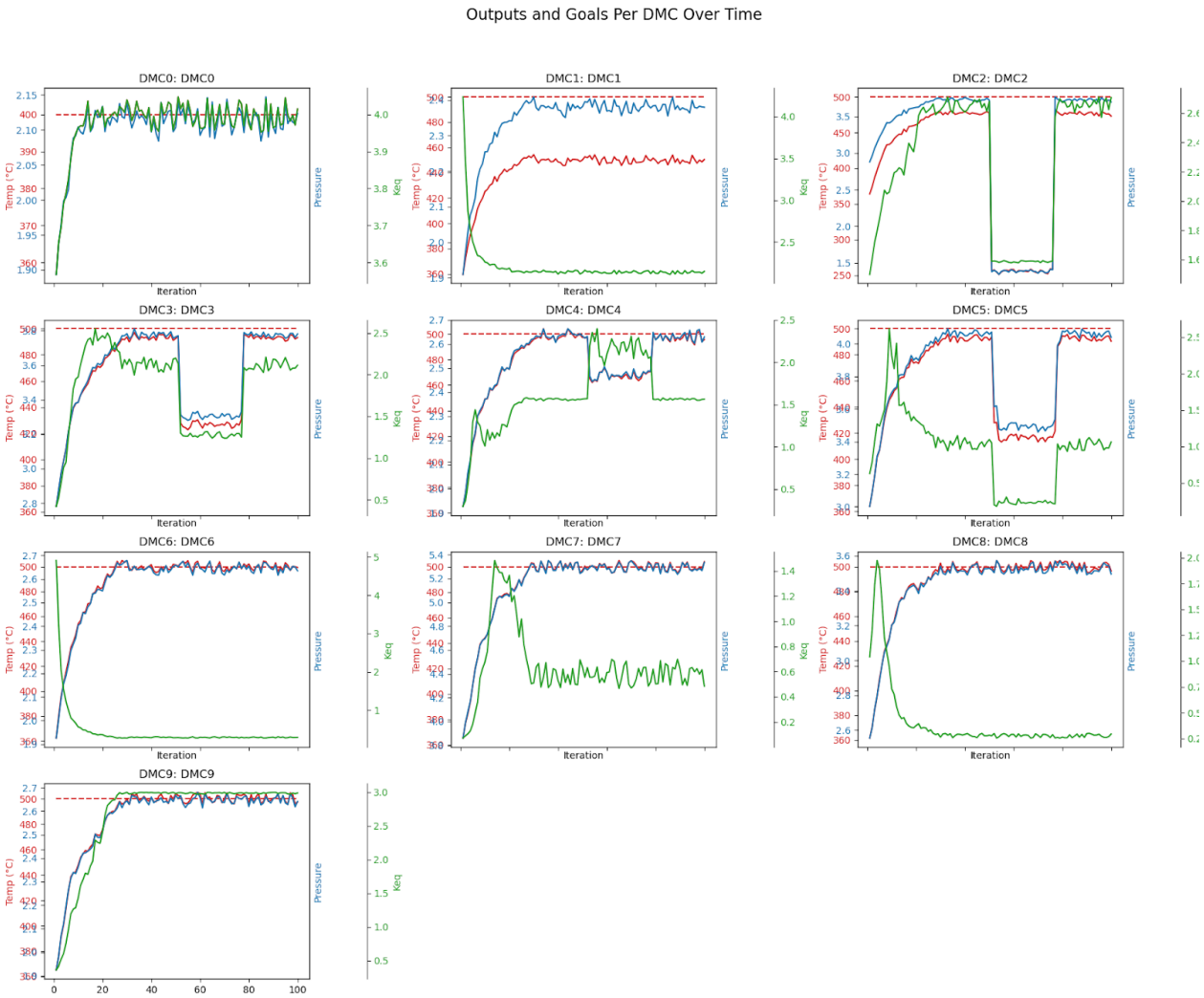


Figure 9: Greatly reduced the goal (and thus the temp) of DMC 9,

Shown in Figure 9, the change in goal of DMC 9 immediately causes a response in 2, 3, and 4 - these are next in the sequence. This demonstrates that our environment continuously responds to changing conditions from an actor.

Reward function

The revised reward function was designed to encourage productive yet safe control behavior by reaching the goal more efficiently. The soft penalty was to increase rapidly when far from constraint bounds, discouraging unsafe control decisions. When the agent moves closer to the

constraint limits, the penalty significantly decreases. This allows the agent to learn safer behavior instead of being abruptly penalized linearly near constraint bounds. The reward function also gives a positive reward for maximizing a high equilibrium constant and a hard, flat penalty for violating bounds.

$$\text{addition}_{hard} = \begin{cases} x \in \text{constraint}, & 2 \\ x \notin \text{constraint}, & -5 \end{cases}$$

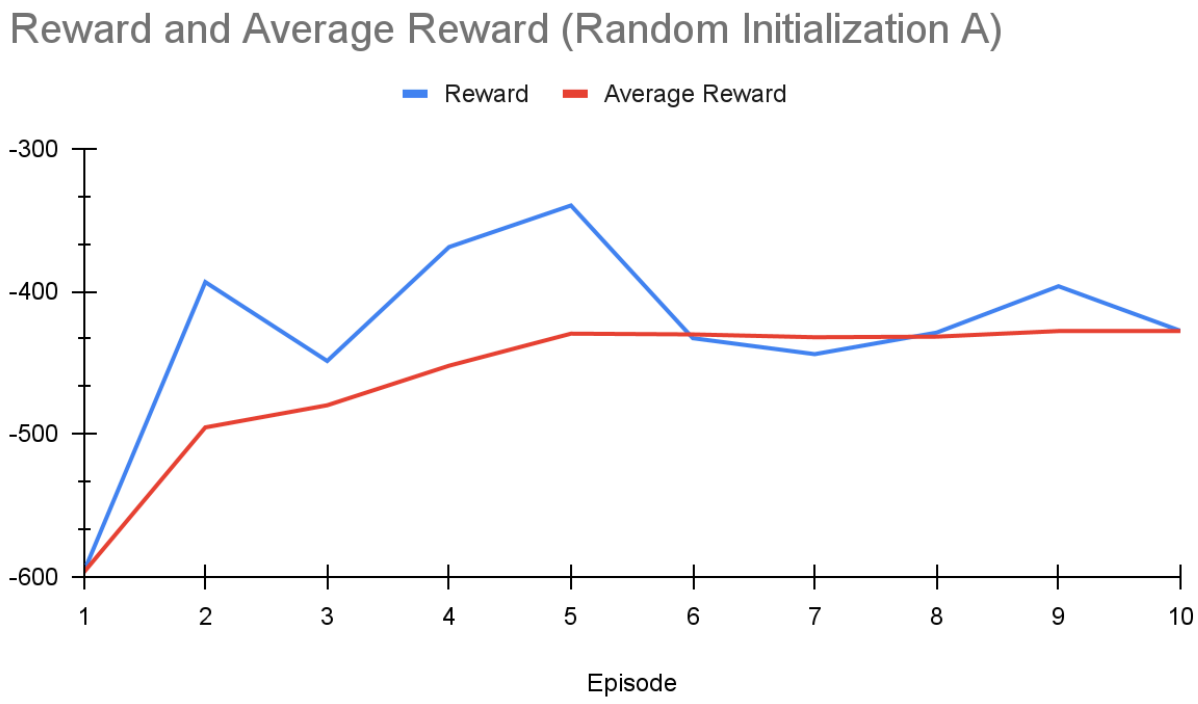
$$\text{penalty}_{soft} = \left(\frac{|x - \text{center}|}{\text{range}}\right)^2$$

$$\text{reward}_{positive} = 100 * K_{eq}$$

$$\text{reward}_{final} = \text{reward}_{positive} + \text{addition}_{hard} + \text{penalty}_{soft}$$

Results

This is a 2000-step SAC model, trained on 10 episodes of 200 steps. The model was converging after 10 episodes, so training was then stopped.



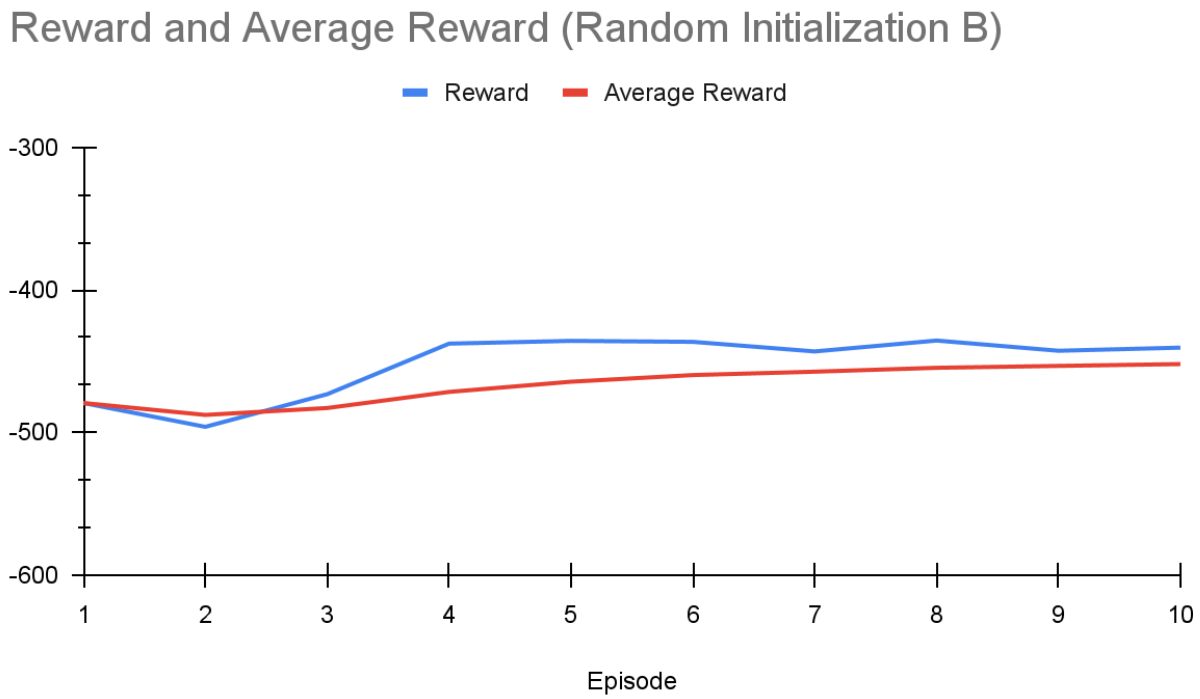


Figure 10: Graph of reward function vs episode.

Our Soft Actor-Critic (SAC) model converges—but not in a satisfying way. While convergence is often considered a positive outcome, our results paint a more nuanced picture. The reward trends remain strongly negative, hovering around -400. The first model plateaus between -420 and -340, while the second one is stuck around -450. Although the policy stabilizes, the results suggest that it's converging to a poor local optimum.

Upon inspecting the learned policy, it appears to be brutally simple. The SAC model consistently outputs extreme values, essentially min-maxing the temperature settings across the system:

```
Action: [-0.999999 -0.999999 -0.999999 -0.999999  0.999999  0.999999
0.999999 0.999999 -0.999999 -0.999999]
```

This behavior makes sense numerically - the outputs are constrained to the range -1,1 due to the tanh activation used in SAC architectures. These values must then be scaled to the physical temperature lower and upper bounds, meaning the SAC is repeatedly choosing actions at the operational extremes.

We found this surprising, since our reward structure rewards high K_{eq} values and penalizes boundary violations. The reward function is working as intended—we confirmed this by running a static scenario in which each DMC's goal was hard-coded to its theoretical maximum value based on the system equations. The resulting reward (Figure 11) was positive and stable, consistent with expectations and confirming the logic of the reward function.

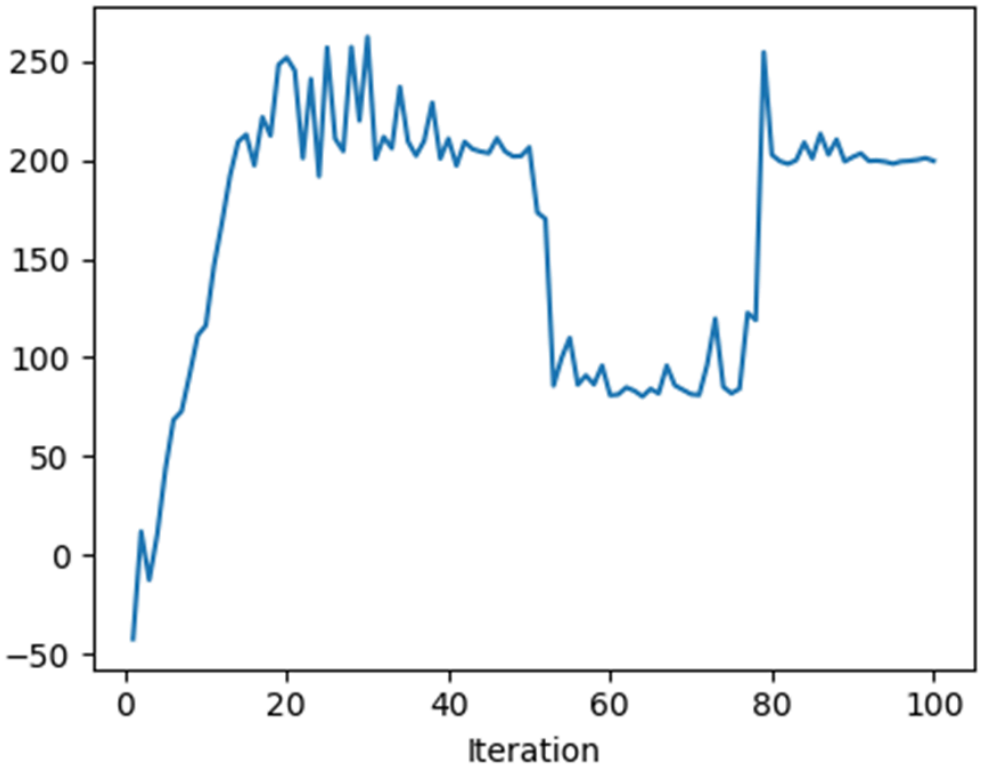


Figure 11: Static DMC run with goal temperatures set to maximize Keq. The resulting reward trajectory is positive and stable, validating the reward function design.

Since SAC encourages exploration, it's normal for the agent to take suboptimal actions early on, especially in a complex, continuous control problem. That said, training is slow. The complexity of the system and the continuous action space make convergence challenging. At this scale, it's clear that meaningful learning will require much longer training, likely on the order of hundreds to thousands of episodes, before stable performance and a well-behaved policy is reached. Further refinement of the reward function will be key to accelerating this process and providing clearer guidance to the agent.

Discussion

The improved model, using Soft Actor-Critic (SAC), is expected to have higher rewards and safer control decisions over time. SAC's exploration is theoretically great for DMC coordination, where decisions are made continuously rather than discretely. SAC allows for dynamic learning, which enables real-time feedback. Over thousands of episodes, we expect the SAC agent to continuously increase its cumulative reward while maintaining DMC temperature and pressure bounds. This allows for smoother and safer operation.

Unlike traditional methods like Hybrid Adaptive DMC, which rely on fixed neural-evolutionary algorithms, SAC enables dynamic decisions through entropy-regularized exploration, making it better suited for coordinating multiple DMCs. SAC also offers major advantages over earlier RL methods like DDPG or PPO by using entropy-regularized exploration, which encourages the agent to balance risk-taking with steadiness. While early SAC performance shows suboptimal results, the infrastructure provides a strong foundation for long-term convergence and stability. We explore a few hypotheses as to why our SAC agent is struggling.

First, the DMC behavior could be flawed. It's worth revisiting how the DMC black boxes handle extreme goals. However, the successful static run suggests the DMC controllers are capable of producing good performance when configured correctly.

The second possibility is that the SAC model is just revealing the ground truth: that for our engineered DMC webs, extreme actions are the most effective for optimizing Keq. This would be consistent with what we saw from the baseline model and might point to inherent limitations or emergent behavior in our DMC design rather than a flaw in SAC itself.

We may also not be training long enough. Our model converges within just 2000 training steps, which is orders of magnitude shorter than typical SAC training durations (200 K–1 M+ steps). So while convergence is impressive, especially after GPU parallelization, the result is likely premature. With extended training, we may see better policies emerge. Even as-is, our current model serves as a proof-of-concept and could be scaled to the full ICE system for deeper evaluation.

Adjusting the learning rate or temperature parameters might help greater exploration and help the model avoid premature convergence. Right now, it seems to settle too quickly, possibly without fully exploring the space of feasible policies.

Given the complexity of continuous control in our system, we're considering a hybrid approach for future directions. Specifically, we propose initially deploying a stateless bandit algorithm to find a baseline optimal configuration for the DMC network. In this model, the agent would treat each decision as independent, assuming a constant input and goal. This simplifies the optimization task and allows us to probe the DMC webs for optimal steady-state actions.

Once the bandit finishes its sweep, we can introduce the SAC model with a head start - either by initializing near the bandit's solution or by training it to replicate and refine the bandit's outputs. SAC would then act as the dynamic controller, adapting to variable conditions and managing continuous control around the near-optimal point.

This two-stage framework provides both computational efficiency and strategic insight, especially when applied to systems as complex and nonlinear as ours.

Moving forward, SAC remains an excellent choice as an RL agent due to its stability, entropy-driven exploration, and suitability for continuous control problems. However, performance may be better by exploring other actor-critic methods such as Twin Delayed DDPG (TD3). Exploring algorithms like Dreamer or World Models may be a possibility for improving faster convergence by improving sample efficiency. Ultimately, SAC shows promising results, given our current DMC complexity, control space, and feedback frequency.

Github link: https://github.com/AnikaP80/proc-control

Sources

[1] wcaarls, “GitHub - wcaarls/grl: Generic Reinforcement Learning Library,” GitHub, 2015. https://github.com/wcaarls/grl (accessed Jan. 31, 2025).

[2] E. M. L. Luz and W. Caarls, “Comparison of reinforcement learning techniques for controlling a CSTR process,” *Brazilian Journal of Chemical Engineering*, Dec. 2023, doi: https://doi.org/10.1007/s43153-023-00422-y.

[3] shakti365, “GitHub - shakti365/grl: Soft Actor-Critic,” GitHub, 2020. https://github.com/shakti365/soft-actor-critic

[4] Georgiou, A., Georgakis, C. and Luyben, W.L. (1988), “Nonlinear dynamic matrix control for high-purity distillation columns,” AIChE J., 34: 1287-1298. https://doi.org/10.1002/aic.690340807

[5] Garcia, Carlos E., David M. Prett, and Manfred Morari. "Model Predictive Control: Theory and Practice—A Survey." *Automatica*, vol. 19, no. 6, 1983, pp. 719–732. Elsevier, https://doi.org/10.1016/0005-1098(83)90002-2.

Individual contributions

Anika	• Dataset:
	  ○ Created DMC framework, including DMC_controller and DMC_structure
	  ○ Created a data generation system (iteration and updating)
	  ■ Implemented recursive updating for each DMC
	  ○ Helped adapt play interface
	  ○ Created complex DMC web + equations
	  ○ Reward function
	  ○ If it's data-generation/simulation-related, I did it

Arian	• Baseline Model:
	  ○ Created baseline reward function
	  ○ Created baseline gym (generate_data() and random sampling)
	  ○ Created baseline naive exploration, training, and optimal value searching
	• Deployment of RL models:
	  ○ Adapted gym interface, play.py (exploration), sac model
	  ○ Rebuilt the environment
	  ○ Wrote everything and fixed everything and ran everything and plotted everything

Kenny	• Helped develop equations for pressure, equilibrium constant, temperature, and volume • Developed DMC infrastructure • Developed overall project vision and idea for ML for DMC • Programmed reward function • Aided in miscellaneous code
Nate	• Helped develop equations for pressure, equilibrium constant, temperature, and volume • Developed DMC infrastructure • Developed overall project vision and idea for ML for DMC

	• Programmed reward function • Introduced DMC topic to team and fundamentals of Process Control