12-770: Assignment #3: Reinforcement Learning for Sustainable Data Center Control

Mario Bergés

2026-03-20

! Clarifications and Setup Guidance

The following notes address common setup issues and clarify aspects of the assignment based on testing. Please read these before starting.

Repository branches:

- **SustainDC (`dc-rl`):** The `main` branch is recommended. The `PyDCM` branch also works but requires Ray, which adds installation complexity. Either branch is acceptable.
- **Gnu-RL:** Use the `pytorch2-compat` branch for PyTorch 2.x compatibility.

Python version: Python 3.10 is required (constrained to `>=3.10, <3.11`). Other versions may cause dependency conflicts.

Dashboard import patch: In `sustaindc_env.py` (line 32), the HARL dashboard import will fail without the full HARL dependency tree. Wrap it in a try/except:

```
# Replace this line in sustaindc_env.py (line 32):
#   from harl.envs.sustaindc.dashboard_v2 import Dashboard
# With:
try:
    from harl.envs.sustaindc.dashboard_v2 import Dashboard
except ImportError:
    Dashboard = None  # Dashboard not available without full HARL deps
```

`dc_config_file` parameter: When calling `make_dc_pyeplus_env()` or setting up `SustainDC`, pass `dc_config_file='dc_config.json'` explicitly. The default parameter value (`dc_config_file.json`) does not match the actual filename.

`month` parameter: This is required in the env config dict — omitting it causes a crash. Use 0-indexed values (0 = January, 11 = December).

Episode termination: SustainDC signals episode end via `truncated["__all__"]`, not `terminated["__all__"]`. Check both flags in your training loop.
`reset()` API: The `SustainDC` wrapper's `reset()` returns only the observation dict, not an `(obs, info)` tuple. Access info via `env.infos` after calling reset.

Setting Up the SustainDC OpenAI Gym Environment (25%)

In this task, you will set up the SustainDC framework and create a simplified control scenario focusing on the Data Center Cooling environment.

Installation and Environment Setup (8%)

Install the SustainDC framework from the GitHub repository at `https://github.com/HewlettPackard/dc-rl` (the BuildSys paper's implementation is in branch `PyDCM`). Follow the installation instructions and ensure all dependencies are properly configured.

Verify your installation by running the provided example scripts and confirming that the OpenAI Gym environments can be instantiated.

Tested Setup Commands

The following setup has been verified to work with `uv` (a fast Python package manager):

```
git clone https://github.com/HewlettPackard/dc-rl.git sustaindc
cd sustaindc
uv init --python 3.10
# Edit pyproject.toml to set: requires-python = ">=3.10, <3.11"
uv add numpy==1.23.5 pandas==1.5.3 gymnasium==0.29.1 scipy==1.13.0 \
       PyYAML==6.0.1 PsychroLib==2.5.0 "setuptools<72" matplotlib \
       torch==2.0.0 dash dash_bootstrap_components
```

After applying the dashboard import patch (see above), verify your installation with:

```python
import sys
sys.path.insert(0, '.')
from sustaindc_env import SustainDC

env_config = {
    'agents': ['agent_dc'],
    'location': 'ny',
    'cintensity_file': 'NYIS_NG_&_avgCI.csv',
    'weather_file': 'USA_NY_New.York-Kennedy.epw',
    'workload_file': 'Alibaba_CPU_Data_Hourly_1.csv',
    'datacenter_capacity_mw': 1,
    'dc_config_file': 'dc_config.json',
    'month': 0,
    'days_per_episode': 30,
}

env = SustainDC(env_config)
obs = env.reset()
obs2, rew, term, trunc, info = env.step({'agent_dc': 1})
print(info['agent_dc'].keys())  # Should show ~48 metrics
```

Both the `main` branch and the `PyDCM` branch are acceptable for this assignment. The `main` branch is simpler to set up.

Configuring a Simplified Scenario (10%)

Using the `dc_config.json` configuration file, create a simplified data center scenario. You can do this by either:

- **Option A**: Configure the Workload and Battery environments to use fixed baseline controllers (you may use the provided baseline implementations)
- **Option B**: Remove the Workload and Battery environments entirely from your configuration

Document your choice and justify it based on computational constraints and learning objectives.

Recommended Approach (Tested)

Use `agents: ['agent_dc']` in the `SustainDC` wrapper configuration. This automatically activates baseline agents for workload and battery — no manual removal or separate configuration is needed. This corresponds to Option A above.
The following `env_config` dict has been tested and works:

```
env_config = {
    'agents': ['agent_dc'],
    'location': 'ny',
    'cintensity_file': 'NYIS_NG_&_avgCI.csv',
    'weather_file': 'USA_NY_New.York-Kennedy.epw',
    'workload_file': 'Alibaba_CPU_Data_Hourly_1.csv',
    'datacenter_capacity_mw': 1,
    'dc_config_file': 'dc_config.json',
    'month': 0,
    'days_per_episode': 30,
}
```

On the data center layout: Modifying the physical layout via `dc_config.json` is not required — the default configuration works well. If you have already customized the layout, that work is perfectly valid and we welcome it in your submission.
Available locations and their files:

Location	`location`	`weather_file`	`cintensity_file`
New York	`'ny'`	`USA_NY_New.York-Kennedy.epw`	`NYIS_NG_&_avgCI.csv`
Arizona	`'az'`	`USA_AZ_Phoenix-Sky.Harbor.epw`	`AZPS_NG_&_avgCI.csv`
Washington	`'wa'`	`USA_WA_Seattle-Tacoma.epw`	`BPAT_NG_&_avgCI.csv`
California	(use `'ny'` sizing)	`USA_CA_San.Jose-Mineta.epw`	`CISO_NG_&_avgCI.csv`
Texas	(use `'az'` sizing)	`USA_TX_Dallas-Fort.Worth.epw`	`ERCO_NG_&_avgCI.csv`
Georgia	(use `'az'` sizing)	`USA_GA_Atlanta-Hartsfield-Jackson.epw`	`SOCO_NG_&_avgCI.csv`
Illinois	(use `'ny'` sizing)	`USA_IL_Chicago.OHare.epw`	`MISO_NG_&_avgCI.csv`

The `location` parameter affects chiller sizing (max ambient temperature assumption). Use `'ny'` for cooler climates, `'az'` for hot climates, and `'wa'` for mild climates.

Configure the Data Center Cooling environment with:

- A reasonable data center layout (e.g., 4-5 rows, 5 racks per row)
- Appropriate HVAC parameters for your location
- Weather data for a location of your choice from the provided options

Baseline Controller Implementation and Testing (7%)

Implement a simple baseline controller for the Data Center Cooling environment. This could be:

- A fixed setpoint controller
- A rule-based controller (e.g., ASHRAE Guideline 36)
- A proportional (P) controller

Run the environment with your baseline controller for a simulation period of at least one week. Record and report:

- Total energy consumption (HVAC and IT)
- Carbon footprint
- Temperature tracking performance (RMSE from setpoint)
- Water usage

Info Dict Reference

The environment provides detailed metrics in the info dict after each step. Key fields for reporting:

Metric	Info Dict Key	Notes
Zone temperature	`dc_int_temperature`	Mean rack outlet temp (°C). Always higher than CRAC setpoint due to supply approach temperature.
CRAC setpoint	`dc_crac_setpoint`	Current cooling setpoint (°C)
HVAC power	`dc_HVAC_total_power_kW`	Total cooling system power

IT equipment power	`dc_ITE_total_power_kW`	Server/compute power
Total DC power	`dc_total_power_kW`	HVAC + IT + auxiliary
Water usage	`dc_water_usage`	Cooling tower water consumption (L per 15-min interval)
Carbon intensity	`norm_CI`	Normalized grid carbon intensity
Carbon footprint	`bat_CO2_footprint`	Cumulative CO_2 footprint
Ambient temperature	`dc_exterior_ambient_temp`	Outside air temperature (°C)
CPU workload	`dc_cpu_workload_fraction`	Server utilization $[0, 1]$

Temperature tracking: "RMSE from setpoint" can be interpreted as either: (1) RMSE between `dc_int_temperature` and a target zone temperature, or (2) RMSE between `dc_crac_setpoint` and its target value. Both interpretations are acceptable — document which you use.
Action mapping: The DC agent's action space is `Discrete(3)`:

- Action 0: decrease setpoint by 1°C
- Action 1: maintain setpoint (do-nothing)
- Action 2: increase setpoint by 1°C

The setpoint is bounded to $[15.0, 21.6]$ °C. The initial setpoint is 18°C.
Timing: Each timestep is 15 minutes. One day = 96 steps. One week = 672 steps. 30 days = 2,880 steps.

Implementing and Evaluating Gnu-RL (40%)

In this task, you will implement the Gnu-RL algorithm with a Differentiable MPC policy and apply it to the SustainDC environment. The Gnu-RL implementation is available at `https://github.com/INFERLab/Gnu-RL`.

Understanding the Differentiable MPC Policy (5%)

Review the Differentiable MPC formulation from the Gnu-RL paper and Lecture 9 notes.

Explain in your own words:

- How the Differentiable MPC policy encodes domain knowledge about planning and system dynamics
- The key components: cost function, constraints, dynamics model, and decision variables
- How the policy differs from a standard neural network policy

Implementing the Linear Dynamics Model (10%)

Implement the linear state-space model for the data center thermal dynamics:

$$x_{t+1} = Ax_t + B_u u_t + B_d d_t$$

Where:

- x_t is the state vector (zone temperatures, etc.)
- u_t is the control action (cooling setpoint)
- d_t is the disturbance vector (weather, workload, etc.)
- A, B_u, and B_d are the model parameters to be learned

Define appropriate state variables, control actions, and disturbances for your scenario.

> **Recommended State Decomposition**
>
> A tested starting point for the state-space decomposition:
>
> - **State** (x_t): zone temperature (`dc_int_temperature`), $n_{\text{state}} = 1$
> - **Control** (u_t): CRAC setpoint delta (change in setpoint), $n_{\text{ctrl}} = 1$
> - **Disturbances** (d_t): ambient temperature (`dc_exterior_ambient_temp`), CPU workload fraction (`dc_cpu_workload_fraction`), ITE power (`dc_ITE_total_power_kW`) — $n_{\text{dist}} = 3$
>
> Setpoint bounds are $[15.0, 21.6]$ °C.
> **Continuous vs. discrete actions:** The Differentiable MPC naturally produces continuous control actions, but the `SustainDC` environment expects `Discrete(3)`. You have two options:
>
> 1. **Discretize the MPC output:** Map continuous actions to discrete using thresholds (e.g., $< -0.33 \to 0$, $> 0.33 \to 2$, otherwise $\to 1$).
> 2. **Bypass discrete actions:** Directly set `dc_env.raw_curr_stpt` to the MPC's continuous setpoint output.
>
> Either approach is acceptable. You are welcome to use a richer state representation if you wish.

Pre-training with Imitation Learning (10%)

Generate expert demonstration data by running your baseline controller for at least 30 days of simulation time (~2,880 timesteps). For this submission, 30 days is sufficient for meaningful dynamics learning. If you have already collected 3 months of data, that additional data will only improve your model — keep it.

> **Critical: Data Collection Policy Must Include Action Variation**
>
> A fixed-setpoint baseline provides zero variation in the control signal, making it impossible for the dynamics model to learn B_u (the control-to-state coefficient). Your data collection policy **must** include some action variation — for example:
>
> - A randomized policy that occasionally increases or decreases the setpoint
> - A rule-based controller that adjusts the setpoint in response to temperature or workload conditions
> - A sinusoidal or scheduled setpoint profile that systematically varies the control input
>
> Without control variation, your learned model will not capture how setpoint changes affect zone temperature, and your MPC policy will not be able to plan effectively.

Implement the imitation learning pre-training procedure:

- Define the loss function that minimizes both state prediction error and action matching error
- Use the hyperparameter λ to balance the importance of states vs. actions
- Train the Differentiable MPC policy parameters (A, B_u, B_d) to match the expert demonstrations

Report your training loss curves and final test performance.

> **Tip**
>
> Normalize your features before training (e.g., min-max scaling to $[0, 1]$). This helps the optimizer converge and prevents features with large magnitudes (like power in kW) from dominating the loss. Store your normalization parameters for reuse during online learning and evaluation.

Online Learning with Policy Gradients (10%)

Implement the online learning phase using Proximal Policy Optimization (PPO) or a similar policy gradient method.

Deploy your pre-trained agent in the SustainDC environment and continue training using a different configuration from pre-training — for example, a different location (which provides different weather and carbon intensity patterns) or a different starting month.

For this submission, at least 7 days of simulation time (~672 timesteps) is sufficient to observe improvement. If you have already run 2 months of online training, that additional experience will strengthen your results.

> MPC Integration Notes
>
> When using the Differentiable MPC as the policy:
>
> - Pass `n_batch=1` to the `mpc.MPC()` constructor — without it, the solver cannot infer the batch size and will throw an error.
> - Map the continuous MPC output to discrete actions for the environment (see Section 2.2 tip above), or set `dc_env.raw_curr_stpt` directly.

Report:

- Learning curves showing reward/cost over time
- Comparison of performance metrics before and after online learning
- Examples of learned behaviors (e.g., pre-heating/pre-cooling strategies)

Multi-Configuration Evaluation (5%)

Evaluate your trained Gnu-RL agent under at least two different configurations — three is ideal if time permits. If you have already evaluated under three or more configurations, that thoroughness will be recognized.

> Available Evaluation Configurations
>
> **Workload traces** (in `data/Workload/`):
>
> - `Alibaba_CPU_Data_Hourly_1.csv`
> - `Alibaba_CPU_Data_Hourly_2.csv`
> - `GoogleClusteData_CPU_Data_Hourly_1.csv`
>
> **Locations:** See the location table in Section 1.2 above. Each location provides distinct weather patterns and carbon intensity profiles.
> **Cost function weights:** Vary the balance between energy minimization and thermal comfort to explore trade-offs.

Compare performance across configurations and discuss:

- Different weather patterns (e.g., different geographic locations or seasons)
- Different workload traces (Alibaba vs. Google)
- Different cost function weights (balancing energy vs. comfort)

Discuss the agent's ability to generalize across these configurations.

Experimental Design and Policy Analysis (35%)

In this task, you will design and conduct an original experiment to investigate an interesting scenario related to sustainable data center control.

Experimental Design (10%)

Propose an experiment that addresses one of the following themes (or propose your own with instructor approval):

Theme 1: Controller Comparison Compare Gnu-RL against at least two other control strategies (e.g., baseline rule-based, standard MPC, model-free RL). Analyze trade-offs in:

- Energy consumption and carbon footprint
- Thermal comfort and constraint satisfaction
- Sample efficiency and training time
- Robustness to model uncertainty

Theme 2: Sustainability Policy Analysis Investigate how different carbon intensity profiles and energy pricing structures affect optimal control strategies. For example:

- Compare performance in locations with different grid carbon intensity patterns
- Analyze the impact of time-of-use pricing on load shifting and cooling strategies
- Evaluate trade-offs between operational cost, energy consumption, and carbon emissions

Theme 3: Forecast Sensitivity Analysis Evaluate how the quality of weather and workload forecasts impacts controller performance:

- Compare perfect forecasts vs. realistic forecast errors
- Analyze degradation in performance as forecast horizon increases
- Investigate robust control strategies under forecast uncertainty

Your experimental design should include:

- Clear research question(s) and hypotheses
- Description of experimental conditions and variables
- Metrics for evaluation
- Number of replications and statistical analysis plan

Experimental Implementation and Data Collection (15%)

Implement your experimental design and collect data. This should include:

- Running controlled experiments with proper random seeds for reproducibility
- Collecting comprehensive performance metrics
- Ensuring sufficient statistical power (multiple runs if needed)

Document any implementation challenges and how you addressed them.

Analysis and Discussion (10%)

Analyze your experimental results and discuss:

- Quantitative findings: Present tables and figures showing key performance metrics
- Statistical significance: Use appropriate tests to validate your findings
- Practical implications: What do your results mean for real-world data center operations?
- Policy implications: How might your findings inform sustainability policies or industry best practices?
- Limitations: What are the limitations of your study and how might they be addressed in future work?

Provide actionable recommendations based on your findings.

References

- Naug, A., Guillen, A., Luna Gutiérrez, R., Gundecha, V., Ghorbanpour, S., Kashyap, L. D., Markovikj, D., Krause, L., Mousavi, S., Ramesh Babu, A., & Sarkar, S. (2023). PyDCM: Custom Data Center Models with Reinforcement Learning for Sustainability. In *Proceedings of the 10th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (BuildSys '23), 299–302. https://doi.org/10.1145/3600100.3623732
- Naug, A., Quinones-Grueiro, M., & Sarkar, S. (2024). SustainDC: Benchmarking for Sustainable Data Center Control. In *Proceedings of the 38th International Conference on Neural Information Processing Systems* (NeurIPS '24), Article 2033. https://doi.org/10.5555/3737916.3741108
- Chen, B., Cai, Z., & Bergés, M. (2019). Gnu-RL: A Precocial Reinforcement Learning Solution for Building HVAC Control Using a Differentiable MPC Policy. In *Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation* (BuildSys '19), 316–325. https://doi.org/10.1145/3360322.3360849