Efficient HVAC Control with Deep Reinforcement Learning and EnergyPlus

Jared Markowitz* **Nathan Drenkow***
Johns Hopkins University Applied Physics Laboratory

Abstract

Heating and cooling comprise a significant fraction of the energy consumed by buildings, which in turn account for a significant fraction of society's energy use. Most building heating, ventilation, and air conditioning (HVAC) systems use standard control schemes that meet basic operating constraints and comfort requirements but with suboptimal efficiency. Deep reinforcement learning (DRL) has shown immense potential for high-performing control in a variety of simulated settings, but has not been widely deployed for real-world control. Here we provide two contributions toward increasing the viability of real-world, DRL-based HVAC control, leveraging the EnergyPlus building simulator. First, we use the new EnergyPlus Python API to implement a first-of-its-kind, purely Python-based EnergyPlus DRL learning framework capable of generalizing to a wide variety of building configurations and weather scenarios. Second, we demonstrate an approach to constrained learning for this setting, removing the requirement to tune reward functions in order to maximize energy efficiency given temperature constraints. We tested our framework on realistic building models of a data center, an office building, and a secondary school. In each case, trained agents maintained temperature control while achieving energy savings relative to standard approaches.

1 Introduction

A key component of efforts to reduce carbon emissions is more intelligent power management. Control systems built around AI have shown significant potential to efficiently manage complex energy distribution systems (1; 2; 3; 4; 5; 6; 7; 8), and are expected to play a large role in the incorporation of renewable energy sources (9).

Here, we focus on the former aspect and in particular on the problem of heating, ventilation, and air conditioning (HVAC) management in industrial buildings. HVAC systems represent a significant fraction of society's energy expenditure each year. In industrial buildings, HVAC systems consume roughly 38% of the energy budget when all equipment (including lighting, computers, utilities, etc.) is taken into account (10). Globally, buildings account for one third of annual electricity consumption, implying that 12% of global consumption may be attributed to HVAC operation in non-residential buildings (10). At such a scale, even modest gains in efficiency could lead to significant reductions in emissions.

Related Work Traditionally, industrial HVAC systems have been controlled by heuristics that are not as robust or as efficient as desired. In recent years, several groups have successfully piloted AI-based controllers in an effort to increase efficiency. Depending on the system and the techniques applied, they have shown energy savings of 10-40% (1; 3; 5; 6; 7; 8; 11). However, these early works demonstrated the viability of Deep Reinforcement Learning (DRL) for HVAC control in only specific, custom settings.

One way to formulate a more general DRL approach to HVAC control is to build around an environment that may be configured for a variety of buildings and settings. In this study

*Equal contribution

we consider EnergyPlus (E+), a widely used, high-fidelity simulator of building physics built by the National Renewable Energy Laboratory (NREL) and others (12). E+ allows for precise specification of building design and construction as well as the consideration of different weather conditions. Prior work (3) demonstrated successful application of DRL to HVAC control by patching E+ to use the Energy Management System (EMS) to pass information between E+ and the DRL agent. This enabled the use of the standard OpenAI Gym workflow (13), but at the expense of non-trivial modifications to building specification files, use of the EnergyPlus Runtime Language (ERL) to specify a building-specific interface for E+, and patching of E+ source code to enable information to flow between the simulator and agent. In releases of E+ version 9 and later, a Python API was introduced. This API allows the user to retrieve and set values of simulation variables, opening up new possibilities for integrating DRL-based control into E+.

Contributions In this work, we provide a Python-based DRL framework for learning HVAC controls for arbitrary building configurations modeled by EnergyPlus. Unlike existing published frameworks, our platform uses the newly released E+ Python API, allowing it to be applied to *any* building for which an EnergyPlus configuration is available while running completely native to Python. We demonstrate energy savings from DRL-based HVAC control (compared to the E+ baseline) on three separate buildings using this framework. We also propose a constrained RL approach for maximizing energy efficiency subject to zone temperature constraints, minimizing the need for reward shaping and providing a "knob" with which to trade comfort for energy efficiency.

2 Methods

2.1 DRL with EnergyPlus

In reinforcement learning, the agent submits actions to the environment, which executes those actions and returns a new state observation and an accompanying reward signal. This loop is typically implemented using the OpenAI Gym interface (13), and was previously achieved for EnergyPlus through the implementation of a custom, ERL-based interface for each building of interest (3). The new E+ Python API provides simplified access to the appropriate system variables and states via Python callbacks during the simulation run. However the new API adds complexity by inverting the runtime process, in that the RL framework can no longer call *into* the environment at each step. Instead E+ must call *out* to the RL framework at specified callback points during the simulation.

To accommodate this shift in process, we implement the following framework. We first define a new `EnergyPlusEnv` class that holds the E+ `EnergyPlusAPI` object and corresponding simulation state, as well as all simulation handles for accessing sensors, actuators, and meter values at runtime. In contrast to more common DRL control flows, which separate the environment and agent, we instead pass an `agent` callback function to the environment when it is created, in order to link the environment to functionality associated with the agent itself (such as the logic for selecting the next action).

At runtime, the learning code creates the `EnergyPlusEnv` object and starts the training loop by handing control to the environment to run the simulation. During execution, the environment instantiates an E+ instance and, using the Python API, starts the simulation via its `api.runtime.run_energyplus` function.

The `agent` function takes the current observation, reward, done flag, and info object as input (similar to the conventional OpenAI Gym interface (13)). Then, deviating from common practice, the agent not only computes the next action, but performs other necessary operations such as updating experience buffers, computing losses, updating networks, and saving model checkpoints. The result is that EnergyPlus is able to continuously run, with the agent making just-in-time decisions while updating its policy.

The `EnergyPlusEnv` class can be easily extended to new buildings by specifying appropriate sensors, actuators, and meters and implementing any custom functionality (e.g., reward functions) associated with that building. Crucially, this can be done without any manual

modification to its original IDF specification. The learning framework is also readily parallelizable, which we did using python's `mpi4py` package. More details on the entire EnergyPlus DRL setup are provided in Appendix A.

3 Reinforcement Learning Approaches

While in principle any DRL algorithm could be inserted into this learning framework, we focus on standard and constrained versions of Trust Region Policy Optimization (TRPO;(14)). In both cases, we make a key change to the standard formulation: while rewards are tabulated at every time step to compute values and advantages, only the time steps where the HVAC is turned on are considered in updating the policy. This allows the agent to ignore decisions that do not impact the system and leads to significantly more stable learning.

Constrained DRL algorithms address Constrained Markov Decision Processes (CMDPs), which have both positive rewards $r(\mathbf{s}, \mathbf{a})$ and costs $c(\mathbf{s}, \mathbf{a})$ defined for each time step, as well as an overall constraint $C(\tau) = F(c(\mathbf{s}_1, \mathbf{a}_1), \ldots, c(\mathbf{s}_T, \mathbf{a}_T))$ defined over the whole trajectory. The associated learning problem is to find

$$\max_{\theta} J_R(\theta) \quad \text{s.t. } J_C(\theta) \leq d, \tag{1}$$

where $J_R(\theta)$ is the objective based on positive reward, $J_C(\theta) = E_{\tau \sim p_\theta(\tau)} C(\tau)$ is the expected cost over trajectories τ sampled in the environment, and d is a fixed threshold. Our constrained approach uses TRPO in Reward Constrained Policy Optimization (RCPO; (15)). The positive reward term J_R reflects energy consumption, while the negative reward term J_C reflects zone temperature excursions outside thermostat setpoints (see Equation 2).

4 Experiments

To validate our framework, we first trained unconstrained TRPO agents on three building types: a two-zone data center, a three-story, 12-zone office building, and a two-story, 40+-zone secondary school. Additional details on these buildings and our training are provided in Appendix B. In all cases, we followed the zone thermostat schedules given by EnergyPlus, having the agent adjust HVAC temperature setpoints and flow rates without changing temperature requirements in any physical space. To ensure agent versatility, we trained on weather scenarios from two-week periods spread over different times of the year from a single region (Colorado). Our experiments on the two-zone data center used the reward function of (3); in all other cases the reward $r(t)$ was given by

$$\begin{aligned} r(t) = -C_E(E_t - E_{E+}) + C_T \sum_{i=1}^{N} \Big(& (T_{i,t} - T_{H,i,t}) \cdot H(T_{H,i,t} - T_{i,t}) + \\ & (T_{C,i,t} - T_{i,t}) \cdot H(T_{i,t} - T_{C,i,t}) \Big). \end{aligned} \tag{2}$$

Here C_E and C_T are normalization constants, E_t is the current power consumption of the agent (for branches under its control), and E_{E+} is the average power consumption of the E+ baseline controller. The current temperatures, heating setpoints, and cooling setpoints for zone $i = 1...N$ are $T_{i,t}$, $T_{H,i,t}$, and $T_{C,i,t}$. The Heaviside step function H allows nonzero temperature contributions only when the temperature of a zone goes outside its heating and cooling setpoints.

The observation spaces of our agents were comprised of current zone temperatures, heating setpoints, and cooling setpoints, as well as the outdoor temperature. Action spaces were multidimensional and continuous, reflecting temperature and air flow rate setpoints. Policies and values were approximated using two-layer MLPs, with Gaussian policy sampling. All experiments were run over three random seeds. For further details on the training, see Appendix C.

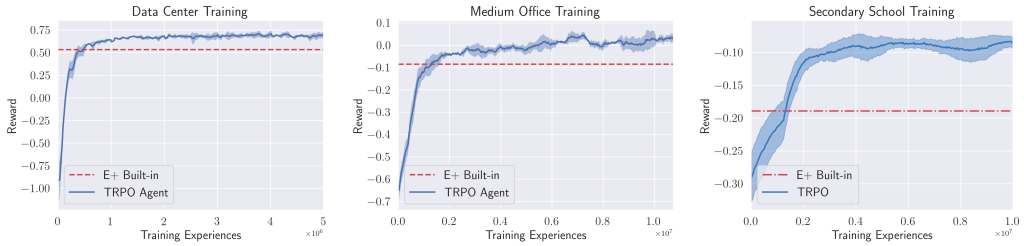

Figure 1: Top row: Unconstrained training of data center agent (left), medium office agent (middle), and secondary school agent (right) using TRPO (14).

Figure 2: Constrained training of agent via Reward-Constrained TRPO (14; 15). The agent learns to respect prescribed temperature bounds (left) while minimizing energy expenditure (right).

Figure 1 shows unconstrained agent learning for three environments, with the performance of the EnergyPlus baseline controller also being plotted for comparison. Figure 2 shows constrained learning for the medium office building. The agent learns to minimize energy while adhering to the temperature constraint. In each case, at least modest energy savings were recorded compared to the EnergyPlus built-in controller- 25.4% in the data center, 3.3% in the school, 7.6% in the office building, and 5.8% for constrained learning of the office building.

Particularly significant gains are achieved in the data center for two reasons. First, it houses IT equipment that is always on and generating heat. This requires the HVAC to work hard nearly constantly to keep the building cool, in contrast to the other two buildings (where the HVAC may be off for long stretches). Second, the IT equipment in the data center operates more efficiently when cool. By including the energy footprint of the IT equipment in its reward function, the data center agent learns to leverage this fact. Such an opportunity is not present in the other building types, leading those agents to record smaller gains.

5 Discussion and Conclusions

These results demonstrate the efficacy of our new learning framework for EnergyPlus, but are only the beginning. To our knowledge, this work is the first fully Python-based framework to enable the application of DRL to *any* building modeled by EnergyPlus. This flexibility should allow for neural-network-based controllers to be evaluated for a broad variety of building layouts and designs, without the need to manually modify building configuration files.

Further, we believe that significantly larger energy efficiency gains can be made through optimized adjustment of thermostat schedules and the presentation of predictive information (e.g., upcoming temperature setpoint changes and predicted weather) to the agent. We intend to study such measures in the future and open source our framework in the hopes that others will as well.

References

[1] Gao, J. Machine learning applications for data center optimization. 2014.

[2] Wei, T., Y. Wang, Q. Zhu. Deep reinforcement learning for building hvac control. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. 2017.

[3] Moriyama, T., G. D. Magistris, M. Tatsubori, et al. Reinforcement learning testbed for power-consumption optimization. In *Asian simulation conference*, pages 45–59. Springer, 2018.

[4] Gamble, C., J. Gao. Safety-first ai for autonomous data centre cooling and industrial control, 2016.

[5] Jia, R., M. Jin, K. Sun, et al. Advanced building control via deep reinforcement learning. *Energy Procedia*, 158:6158–6163, 2019.

[6] Li, Y., Y. Wen, D. Tao, et al. Transforming cooling optimization for green data center via deep reinforcement learning. *IEEE transactions on cybernetics*, 50(5):2002–2013, 2019.

[7] Zou, Z., X. Yu, S. Ergan. Towards optimal control of air handling units using deep reinforcement learning and recurrent neural network. *Building and Environment*, 168:106535, 2020.

[8] Ding, X., W. Du, A. Cerpa. Octopus: Deep reinforcement learning for holistic smart building control. In *Proceedings of the 6th ACM international conference on systems for energy-efficient buildings, cities, and transportation*, pages 326–335. 2019.

[9] Rolnick, D., P. L. Donti, L. H. Kaack, et al. Tackling climate change with machine learning. *ACM Computing Surveys (CSUR)*, 55(2):1–96, 2022.

[10] González-Torres, M., L. Pérez-Lombard, J. F. Coronel, et al. A review on buildings energy information: Trends, end-uses, fuels and drivers. *Energy Reports*, 8:626–637, 2022.

[11] Evans, R., J. Gao. Deepmind ai reduces google data centre cooling bill by 40%, 2016.

[12] Crawley, D. B., C. O. Pedersen, L. K. Lawrie, et al. Energyplus: Energy simulation program. *ASHRAE Journal*, 42:49–56, 2000.

[13] Brockman, G., V. Cheung, L. Pettersson, et al. Openai gym. *CoRR*, abs/1606.01540, 2016.

[14] Schulman, J., S. Levine, P. Moritz, et al. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. 2017.

[15] Tessler, C., D. J. Mankowitz, S. Mannor. Reward constrained policy optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*. 2019.

A DRL with EnergyPlus - Detailed

We provide here additional pseudocode to illustrate how the callback structure of the EnergyPlus Python API can be leveraged to perform RL.

Environment The environment object primarily takes a list of sensors, actuators, and meters as input. Individual sensors in the list are specified as a tuple `(name, key, request_flag)` where `name` and `key` are determined from the list of available variables in the associated building's input data file (IDF). The `request_flag` is set to `True` whenever the variable is not provided by E+ automatically. Meters are specified by a `(name, key)` tuple and actuators by a `(component_type, control_type, key)` tuple. These tuples provide the necessary input for the associated E+ API calls, such as `api.exchange.get_variable_value` or `api.exchange.set_actuator_value`, which will be necessary for allowing the DRL agent to sense and override E+ states.

This setup allows for much more flexible experimentation compared to prior methods, since different sensing/actuation configurations can be changed programmatically or via simpler code definition. Given the uniqueness of different building designs, learning objectives, and observation/action spaces, any new environment must at a minimum implement its own reward, `format_obs()`, and `format_act()` methods.

The `agent_handler` method is an `EnergyPlusEnv` class method that collects sensor, actuator, and meter values via API calls into E+. These observations are formatted using the building environment's `format_obs(raw_obs)` method and then passed to the agent. The `agent` function is a class method of the DRL algorithm whose function handle is passed to the environment at instantiation. The advantage of this approach is that the `agent` function is bound to the DRL agent class and has access to other attributes and methods inherent to the specific learning approach. While this agent-environment relationship is seemingly circular, this approach allows the environment to control and interface with E+ while still enabling the DRL agent's functionality to be distinct and separate from the environment.

Since our objective is to learn efficient HVAC control policies, we rely on the `api.runtime.callback_after_predictor_after_hvac_managers(state, agent_handler)` callback function to interrupt E+ at a point where sensor readings can be obtained and actuator values can be overridden according to the agent's current policy. Using this callback follows prior common practice, which inserted EMS programs after predictors and setpoint managers to recompute relevant actuator values. Care must be taken to ensure that over-writing setpoints at this point in the simulation does not violate interdependencies between setpoints (e.g., where setpoint A is overriden but setpoint B was previously determined by the original value of A). In our case, the RL agent's actions can safely override the Mixed Air setpoint manager values, since they are among the last to be computed in EnergyPlus.

```
# Define the building environment
class Building(EnergyPlusEnv):
    def __init__(agent, sensors, actuators, meters):
        self.agent = agent
        # Get handles to sensors, actuators, meters
        # Setup E+

    def run(self):
        self.api = EnergyPlusAPI()
        self.state = self.api.state_manager.new_state()
        # ... Setup E+ environment ...
        self.api.runtime.run_energyplus(self.state, argv)

    def agent_handler(self, state):
        # Retrieve E+ state information
        reward = self.reward_func(raw_obs)
        obs = self.format_obs(raw_obs)
        raw_action = self.agent(obs, reward, done, info)
        action = self.format_action(raw_action)
        # Set actuators according to action
```

Agent The agent's `callback_train` method is passed to the environment and is the entry point into the agent's primary functionality. When the agent receives the current `observation`, `reward`, `done`, and `info` inputs, it must not only choose the next action, but also perform other necessary updates and computations, such as computing losses and updating policy/-value networks. This `callback_train` method is accessed within the `agent_handler` of the `EnergyPlusEnv`.

```
# Define the learning agent
class DrlAgent:
    def train(self):
        ...
        self.env = Building(agent=self.callback_train, sensors=...)
        while training_steps < max_training_steps:
            self.env.run()

    def callback_train(obs, reward, done, info):
        # Update buffers
        # Compute losses
        # Update/save model parameters
        # Compute next action
        return action
```

Runtime At runtime, the learning code creates the `EnergyPlusEnv` object and starts the training loop by handing control to the environment to run the simulation. During execution, the environment instantiates an E+ instance and, using the E+ Python API, starts the simulation via `api.runtime.run_energyplus`.

```
# ... run script ...
agent_obj = DrlAgent(config)
agent_obj.train()
```

B Buildings

We ran our experiments using the IDF files of three commercial reference buildings from the EnergyPlus 9.6 distribution. Specifically, we evaluated on the Two-Zone Data Center with Economizer, Medium Office, and Secondary School. All buildings used Variable Air Volume (VAV) systems with temperature and flow rate setpoints available for each component of these branches.

Two-Zone Data Center We implemented our DRL-based controls for both zones of the data center similar to (3). We overwrote the EnergyPlus setpoint managers for the single zone cooling components by adding actuators for each of the components of the West and East zone branches. We controlled each component via a temperature and mass air flow rate provided by the DRL agent's policy. The reward was based on adherence to the desired zone temperature setpoint ($23.5°C$), using the temperature reward defined in (3) plus the total facility energy.

Medium Office Building The medium office reference building contains three floors, each with four zones. Air conditions on each floor are controlled through a single VAV system connected to an air source as well as heating and cooling elements. The temperature setpoint, flow rate, and humidity setpoint of each component may be controlled, though we only considered temperature and flow rate.

In contrast to the data center (but similarly to the school), we needed to be able to heat, cool, and allow the fans to turn off. The temperature schedule of the zones was considered to be fixed by EnergyPlus and reflected the standard work week. We allowed the built-in controller to determine when the HVAC system turns on, but had the learned controller determine all temperature and flow setpoints when the system was on.

Secondary School The secondary school consists of four main branches for the main classrooms, offices, lobby, and library covering the first and second floors, with additional systems for the gym, auditorium, cafeteria, and kitchens. The four main branches are variable air volume mixed air systems, each with separate water-based heating and cooling coils. As in the other building systems, we inserted EMS actuators for the temperature and mass air flow setpoints, which were controlled via the DRL agent's policy. These setpoints overrode the Mixed Air setpoints initially computed by EnergyPlus.

Since we did not learn controls for the unitary air systems in the other branches, the energy term in the reward was based on the sum of the electrical energy measured for each branch we controlled. This was found to produce more stable training by allowing the agent to receive feedback more directly related to its policy. Note that by not controlling the unitary air systems, we may not be realizing all possible efficiency gains and our results may be further improved.

C Training Hyperparameters and Architectures

We used the training hyperparameters for TRPO described in Table 1.

Hyperparameter	**Value**
Batch Size	10000
Discount (γ)	0.99
Generalized Advantage Estimate (GAE) (λ)	0.98
Value function learn rate	0.001
Value function training iterations	10
Max KL-divergence	0.01

Table 1: Training Hyperparameters

Note that all training runs were conducted using 20 cores, each collecting 500 experiences.

The value function and policy network architecture details are described in Table 2. All networks in all experiments were simple two-layer Multi-Layer Perceptrons (MLP). The observation and action dimensions differ for each building and those inputs/outputs are described in the prior sections.

Parameter	**Value**
Hidden layer dimensions	$[64, 64]$
Activation function	`tanh`
Min $\log \sigma$	-20
Max $\log \sigma$	2

Table 2: Value function and policy network setup.