DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof. Reference herein to any social initiative (including but not limited to Diversity, Equity, and Inclusion (DEI); Community Benefits Plans (CBP); Justice 40; etc.) is made by the Author independent of any current requirement by the United States Government and does not constitute or imply endorsement, recommendation, or support by the United States Government or any agency thereof.

Flexible Integration of Diverse HVAC Technologies in EnergyPlus via Python-Enabled Workflows

Eric Bonnema, Amy Allen, Edwin Lee, Matt S. Mitchell, and Ryan Meyer

National Renewable Energy Laboratory

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated under Contract No. DE-AC36-08GO28308

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Technical Report
NREL/TP-5500-85628
May 2025

Flexible Integration of Diverse HVAC Technologies in EnergyPlus via Python-Enabled Workflows

Eric Bonnema, Amy Allen, Edwin Lee, Matt S. Mitchell, and Ryan Meyer

National Renewable Energy Laboratory

Suggested Citation
Bonnema, Eric, Amy Allen, Edwin Lee, Matt S. Mitchell, and Ryan Meyer. 2025. *Flexible Integration of Diverse HVAC Technologies in EnergyPlus via Python-Enabled Workflows*. Golden, CO: National Renewable Energy Laboratory. NREL/TP-5500-85628. https://www.nrel.gov/docs/fy25osti/85628.pdf.

NREL is a national laboratory of the U.S. Department of Energy
Office of Energy Efficiency & Renewable Energy
Operated under Contract No. DE-AC36-08GO28308

This report is available at no cost from the National Renewable Energy Laboratory (NREL) at www.nrel.gov/publications.

Technical Report
NREL/TP-5500-85628
May 2025

National Renewable Energy Laboratory
15013 Denver West Parkway
Golden, CO 80401
303-275-3000 • www.nrel.gov

NOTICE

This work was authored by the National Renewable Energy Laboratory for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308 Funding provided in part by the U.S. Department of Energy Office of Energy Efficiency and Renewable Energy Building Technologies Office. The electric vehicle charging case study was funded in part by the National Renewable Energy Laboratory's lab-directed research and development. The refrigeration case study was funded in part by Copeland. The outdoor air pretreatment case study was funded in part by the U.S. General Services Administration. The groundwater heat pump case study was funded in part by the Wells Fargo Innovation Incubator. The views expressed herein do not necessarily represent the views of the DOE or the U.S. Government.

Cover photos (clockwise from left): Josh Bauer, NREL 61725; Visualization from the NREL Insight Center; Getty-181828180; Agata Bogucka, NREL 91683; Dennis Schroeder, NREL 51331; Werner Slocum, NREL 67842.

List of Acronyms

API	application programming interface
BCVTB	Building Controls Virtual Test Bed
DOE	U.S. Department of Energy
EMS	Energy Management System
ERL	EnergyPlus Runtime Language
EV	electric vehicle
EVSE	electric vehicle supply equipment
GWHE	groundwater heat exchanger
HVAC	heating, ventilating, and air conditioning
OA	outdoor air

Executive Summary

Analysis of advanced controls and novel system types is often not directly feasible in building energy simulation tools. Various techniques extend building energy simulation tool capabilities to allow the use of user-defined scripts and programs, but these approaches have limitations. The EnergyPlus™ Python plugin offers users new flexibility to use EnergyPlus to call an external Python module at specific points in the simulation, as well as to use Python to call EnergyPlus functionality through an application programming interface (API). This paper presents four case studies leveraging the EnergyPlus Python plugin to facilitate analysis of advanced controls and system types. The use of the Python plugin offers greater modularity and flexibility relative to previous approaches, is less error prone, and is simpler for users to adopt.

The Python plugin allows EnergyPlus to be used in a more flexible manner and to accommodate the expanding realm of energy modeling applications.

Table of Contents

List of Figures

1 Introduction

The development of building energy simulation programs began in the 1960s. Most notably, DOE-2 (Winkelmann et al. 1993) and BLAST (Building Systems Laboratory 1999) were developed independently utilizing different approaches. However, in 1999, a group of researchers came together to develop a unified building energy simulation program named EnergyPlus (Crawley et al. 2001). At the time, the program represented a major advancement in the development of building energy simulation programs by coupling building heat load calculations with mechanical systems simulation capabilities. The result is a fully coupled energy simulation of the building structure and all mechanical and heating, ventilating, and air-conditioning (HVAC) equipment. As of this writing, updated versions of EnergyPlus are regularly provided, incorporating new features, bug fixes, and general software enhancements.

As EnergyPlus matured and gained adoption, users began requesting the ability to implement user-designed control algorithms, analogous to those deployed in real building control systems for energy management. To fill this need, researchers developed the Energy Management System (EMS). This new feature was originally conceived with core runtime language functionality developed but not exposed to public users through EnergyPlus' input structures (Ellis, Torcellini, and Crawley 2007). The first functional version was released to the public in 2009 in EnergyPlus Version 4.0 (Lawrie 2009; Griffith 2023).

EMS capabilities were developed as a way for users to get data during the simulation from "sensors," and then, using IF-ELSE-THEN control logic, implement control algorithms to change the state of controllable variables within the program, referred to as "actuators." This capability opened the door for users to write their own control logic to make decisions in real time during the simulation to achieve desired outcomes. In the initial implementation, users had the ability to customize relatively simple programmable logic, such as manipulating thermostat set points or equipment and system availability schedules. As EnergyPlus continued to mature, additional sensors and actuators continued to be added, unlocking new opportunities for users to customize control of the simulation.

One limitation of this initial implementation of EMS was that users were required to program the control logic directly into the input file in a rudimentary programming language, referred to as EnergyPlus Runtime Language (ERL). This language supported the following capabilities: "subroutines," which are a way for users to encapsulate smaller blocks of code; IF-ELSE-THEN logic; the naming of variables; and CALL statements to call other ERL subroutines and programs. The programs had little debugging support other than large flat text output files that output records for each line of ERL code that was executed. Despite these difficulties, users were still able to implement novel control algorithms to serve various purposes. Recent studies include Sardoueinasab, Yin, and O'Neal (2018), which used EMS to study the impact of primary air leakage from parallel fan-powered terminal air units, and Goia, Chaudhary, and Fantucci (2018), which used EMS to study the effects of thermal hysteresis in phase change materials.

Another advancement that allows users to interact with and control EnergyPlus simulations was the development of the Building Controls Virtual Test Bed (BCVTB) (Wetter 2011), which was intended to connect various simulation programs (including EnergyPlus, Modelica, Radiance, and MATLAB/Simulink) together with a common data exchange framework. During the

development of BCVTB, an external interface for EnergyPlus was developed to allow for interaction with EnergyPlus data in real time during the simulation. For example, the BCVTB allows one to simulate a building envelope and loads in EnergyPlus and an HVAC and control system in Modelica, which can provide greater flexibility. The program Ptolemy II (University of California 2023) was used as the program orchestrator, meaning that it oversaw the marshalling of data between the various simulation programs.

Building on the work enabled by the BCVTB, MLE+ was developed to extend the capabilities of EnergyPlus by leveraging the previously mentioned external interface to integrate with MATLAB and Simulink (Bernal et al. 2012). MLE+ seeks to facilitate optimization, hardware-in-the-loop analyses, and modeling of advanced control strategies in an EnergyPlus building model, including model predictive control. Sophisticated controls strategies such as model predictive control cannot be readily implemented in EMS because of the rudimentary programming capabilities of ERL. MLE+ offers full access to MATLAB's code generation and debugging capabilities and offers users the ability to externally modify a control scheme without changing the EnergyPlus input file. For example, MLE+ has been used to implement a model predictive control approach for electric radiant floor heating systems in EnergyPlus to minimize system electrical demand, with MLE+ feeding the optimum power levels to EnergyPlus at each time step (Bernal et al. 2012).

Co-simulation, in which multiple modeling tools are used to simulate the performance of a coupled system, is an advanced energy modeling use case (Taveres-Cachat et al. 2021). Examples of systems that can benefit from a co-simulation approach include advanced building facades and daylighting analyses, because of the intersecting physical domains involved in their operation, and the existence of specialized tools focused on modeling their performance, separate from whole-building energy simulation. Co-simulation with a whole-building energy modeling tool offers the opportunity to analyze the performance of these systems in a more nuanced and rigorous way. Co-simulation can also facilitate analysis of advanced control strategies and integration of external data sources into a model. In this way, co-simulation can enhance the flexibility of energy modeling tools through the integration of external modules for modeling particular systems (Taveres-Cachat et al. 2021).

Co-simulation generally involves the exchange of data between tools at each simulation time step, or possibly other frequencies. Because many energy simulation tools, including EnergyPlus, were originally designed to be all-encompassing and used independently of other tools, data exchange between whole-building energy simulation tools and customized models of subsystems is often challenging (Taveres-Cachat et al. 2021). The EnergyPlus API and Python plugin address this challenge by facilitating data exchange between EnergyPlus and external tools.

2 Development of the Python Plugin

Since the original release of EnergyPlus decades ago, the tool has been utilized as a closed box, with the only "interface" being the user's input file(s) and the set of output files generated by the simulation engine. While the simulation was running, it was expected to remain uninterrupted from start to end. Over the years, it became apparent that industry and researchers alike could benefit from additional "simulation time" runtime capabilities. One such capability was user-defined functionality, where the previously described EMS feature was added to the simulation engine. This unlocked enormous capability, which has been adopted by users to provide custom control and custom physical simulation of novel components or configurations. One limitation of the EMS feature is that the logic and data must be entirely defined before the simulation is run. The "external interface" was later added to the simulation engine to provide connection with other tools. Unfortunately, the complexity of this configuration led to minimal adoption.

With the release of EnergyPlus 9.3, two new ways to connect to EnergyPlus were deployed: a "plugin" approach and a "library" approach. Figure 1 depicts the possible connections.

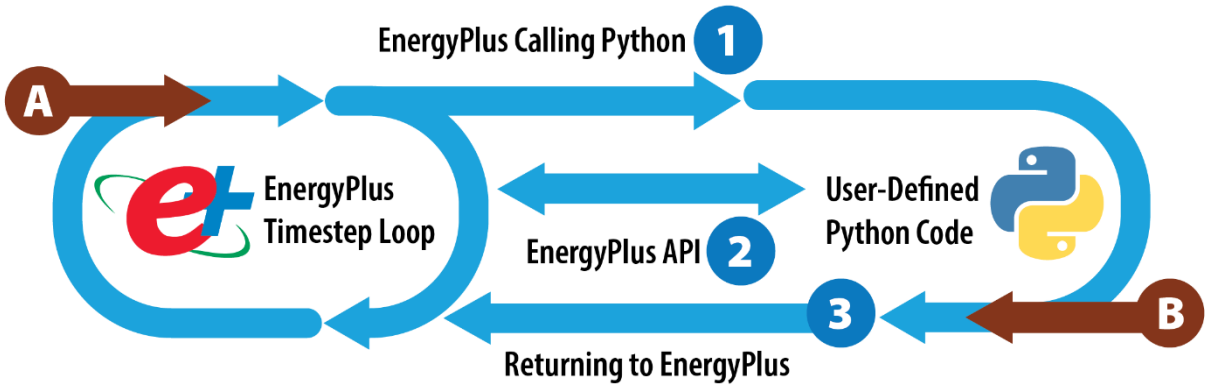

Figure 1. Flow diagram showing available EnergyPlus + Python interfaces; A and B represent workflow entry points for the "plugin" approach and "library" approach, respectively.

To maintain a "traditional" simulation workflow, where EnergyPlus is the driving process, a Python interpreter was embedded inside EnergyPlus. In this "Python plugin" approach, the user writes custom Python code, and the workflow is initiated by providing a path to a Python file in the base EnergyPlus input file and asking EnergyPlus to run. In Figure 1, this is represented by Point A, where the process begins with EnergyPlus. In this workflow, the user's code will be called automatically by EnergyPlus while it is running, and the frequency of the calls is dependent on which Python EMS function is overwritten in the user code.

To enable more flexible workflows, the functionality within the EnergyPlus engine was exposed through a C application programming interface (API) with Python bindings. Users can leverage this interface by calling EnergyPlus as a "library." In Figure 1, this is represented by Point B, where the process begins with a user's Python interpreter. In this workflow, the user's Python code will utilize the runtime API functions to connect to the simulation while it is running, and the frequency of the calls is dependent on which callback functions are used to register user code. Co-simulation with other simulation tools is enabled using these callback functions to synchronize the tool integration times. Communication with EnergyPlus can take place through

the raw API, or by wrapping the API with a Functional Mock-up Interface layer, which is what Spawn of EnergyPlus utilizes to co-simulate Modelica with EnergyPlus (Wetter et al. 2022). When running EnergyPlus as a library, a language like Python can be in control of the software stack, making it easier to install and communicate with dependencies.

The Python plugin approach allows a traditional workflow, which is beneficial for long-term users of EnergyPlus, while the API allows diverse new workflows. Regardless of the entry point into the simulation, the Python code must be able to interact with the simulation. In Figure 1, Point 1 represents the EnergyPlus time integration loop stopping to make a call to user Python code. While inside Python, Point 2 represents the user code calling the EnergyPlus API functions for two primary purposes:

- Property evaluation: The internal psychrometric, steam, and water property calculations can be utilized to perform user-defined physical calculations.
- Data exchange: Sensor data from EnergyPlus (output variables and meters) can be read, and actuators can be written (control signals).

Finally, Point 3 in Figure 1 represents Python returning to EnergyPlus, where the time integration will continue until the next Python calling point is reached. The existing EMS calling points are available for use by the Python plugin (DOE 2023a).

Since the release of the new API and Python EMS, researchers and users have begun exploring the possibilities of connecting EnergyPlus to user-defined code, as well as outside data sources and simulation engines (NREL 2022b; Stripp, Turrin, and Bokel 2024). For example, these applications include:

- Connecting EnergyPlus to weather data sources to allow for complex forecast-based control strategies.
- Connecting EnergyPlus to hardware, treating EnergyPlus as a digital twin to evaluate control strategy options and control the actual building systems.
- Connecting EnergyPlus to trained machine-learning algorithms to determine optimized control signals.
- Using the API to connect EnergyPlus to other languages, such as Modelica, to perform more detailed analysis of parts of the simulation while allowing EnergyPlus to run the whole-building aspects of the simulation and get real-time feedback from interaction.

The new API and plugin system are not adding any new actuators to the simulation. The actuators available in the simulation are carefully chosen to allow a user to modify control signals and operation, while not breaking the energy balance calculations being performed. There is much potential for new actuators to be added into the EnergyPlus code itself, and the development team continually receives requests for new actuators. The purpose of the API and plugin are to make using the existing sensors and actuators more flexible and powerful. By using generic actuators like schedule inputs, a user can leverage Python and countless libraries to manipulate the running simulation.

3 Case Studies

This report presents four case studies describing applications of the EnergyPlus Python plugin. Each case study is meant to be an illustrative example of a way that the Python plugin could be used, rather than to draw meaningful conclusions from simulation results. The cases studies are all examples that would have been exceedingly difficult, if not impossible, to implement using traditional EMS. The case studies address different applications of the Python plugin, including enhancements to existing EnergyPlus capabilities, incorporation of equipment performance data, and modeling of systems for which component models are not available in EnergyPlus, and whose performance is described with complex equations. The use of the Python plugin also enhances the modularity of the modeling workflows, readily enabling future additions and refinements.

3.1 Electric Vehicle Charging

This case study presents an illustrative example of extending EnergyPlus capabilities with a Python model. The Python model is not described in detail, nor are any conclusions presented. Instead, we show how the Python plugin can be used to quickly add a model to EnergyPlus, without the time-consuming process of altering the EnergyPlus code base.

3.1.1 Background

When creating a building energy model, a user may wish to model an electrical or fuel load that consumes energy from a building's infrastructure but does not contribute to the building thermal load. A building may contain a component that is external to the building envelope, but still contributes to a building's utility bill, and correctly capturing a building's utility costs could be a requirement for a building energy model. Some examples include outdoor living items (e.g., natural gas grill or fireplace, televisions, other electronics), outdoor pool pumps and heaters, and electric vehicles (EVs). EVs are becoming increasingly commonplace (International Energy Agency 2020), and managing the electricity needed to charge these vehicles is an area of interest (Gilleran et al. 2021; Muratori et al. 2019). EVs can place a significant electric load on a building, and modeling this EV load in conjunction with the rest of the building load helps inform how a building with an EV may impact the electric grid.

The National Renewable Energy Laboratory has developed a Python model (Mishra et al. 2022) that includes the charging and discharging behavior of an EV as well as the electric vehicle supply equipment (EVSE) (i.e., the EV charging port). This model, dubbed the PyChargeModel, can be configured to generate multiple EV and EVSE agents that can interact with one another. The EnergyPlus Python plugin can be used to instantiate these EV/EVSE agents and model their charging behavior from the perspective of a building electric meter. For example, a single-family home EnergyPlus building model could be coupled with a PyChargeModel for a single charging port to capture how a home-based EV would not only impact that home's electric bill, but also how that home's electric load changed from the utility provider's perspective. Similarly, an EnergyPlus office building model could be connected to multiple PyChargeModels to explore how adding multiple EV chargers to a commercial building would impact the building's electric demand.

3.1.2 Approach

To join an EnergyPlus model with the PyChargeModel, the built-in *Exterior:FuelEquipment* object is used. This object is meant to model an item exterior to the building load considerations to facilitate the reporting of all consuming devices behind a building's utility meter. Here, a placeholder *Exterior:FuelEquipment* object is added to the EnergyPlus model. One of the PyChargeModel outputs is the power draw (in watts) from the charging port. This value is passed to EnergyPlus using an actuator on the schedule for the placeholder *Exterior:FuelEquipment* object. At the beginning of an EnergyPlus simulation, the PyChargeModel is initialized. As the EnergyPlus simulation runs, it tracks time and passes this time to the PyChargeModel. The PyChargeModel executes various EV charging events based on this time. At every EnergyPlus time step, the wattage output from the PyChargeModel is passed to EnergyPlus through the schedule actuator on the placeholder *Exterior:FuelEquipment* object. When the EnergyPlus simulation is complete, EnergyPlus processes the energy consumed by the placeholder *Exterior:FuelEquipment* object(s) as it would any other object, and this energy is included with the standard EnergyPlus report, such as annual electricity consumption or peak electricity demand.

Figure 2 shows the data exchange between EnergyPlus and Python for EV modeling.

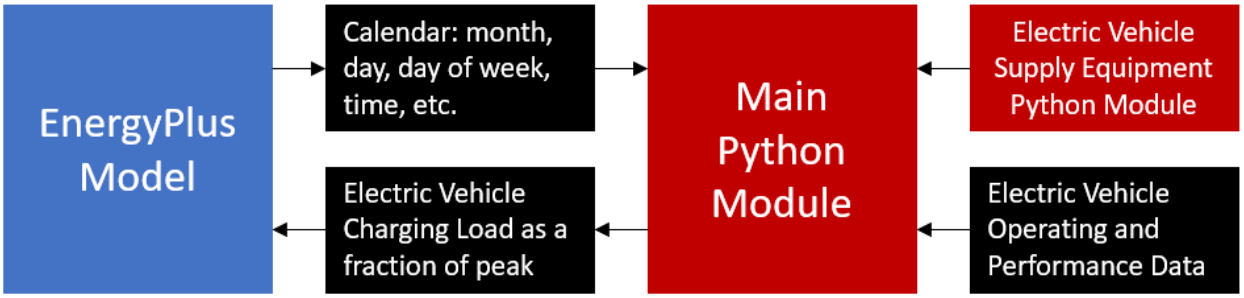

Figure 2. Schematic diagram of EV Python plugin modeling workflow

The PyChargeModel is an independent, dynamic model with its own EV charging objectives. A main reason to use the PyChargeModel instead of a predetermined charging schedule is that the model is "stand-alone"; i.e., it runs independently of the EnergyPlus simulations. In this example, the PyChargeModel was used to simulate the interruption of EV charging as a building demand response measure. Figure 3 shows sample EnergyPlus output with the EV charging model integrated into a building model via the Python plugin. The blue line in Figure 3 shows a week of EV charging in an uncontrolled scenario, and the orange line shows the EV charging power with EV charging being interrupted as a demand response measure.

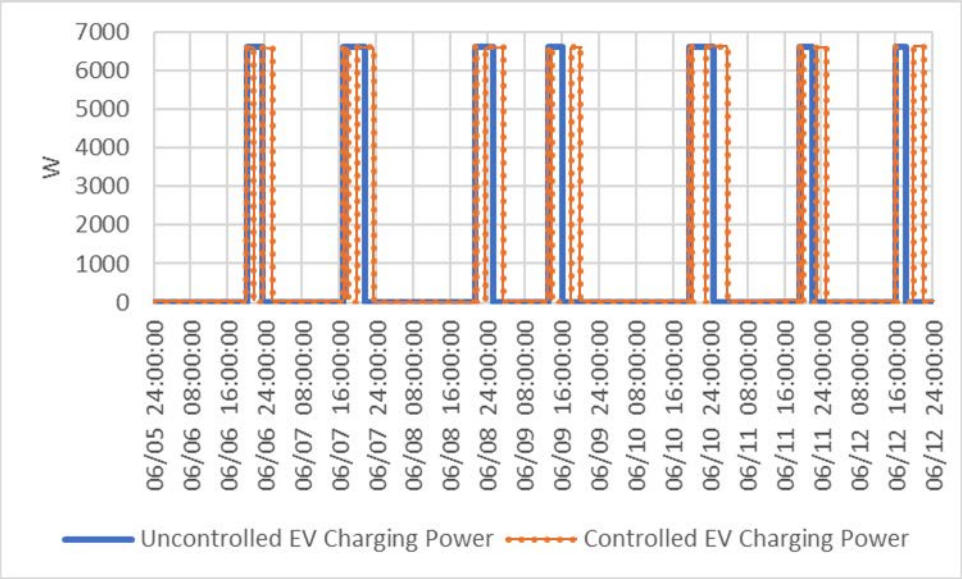

Figure 3. Example EnergyPlus model outputs with Python plugin EV charging

When EV charging is interrupted, the EV retains its current charge state and waits until charging is available again to resume. This type of dynamic EV modeling is made possible with the Python plugin. Traditional EnergyPlus EMS can be used to actuate the *Exterior:FuelEquipment* object schedules. However, co-simulation with the PyChargeModel, providing the EV charge state and power draw of the charger, is only possible with the Python plugin—specifically the connection with the dynamic model that resumes EV charging if it is disrupted.

3.2 Refrigerated Case Modeling

This case study presents an illustrative example of enhancing EnergyPlus' capabilities with a gray box Python model. The gray box model is not described in detail, nor are any conclusions presented. Instead, we show how the Python plugin can be used to incorporate additional considerations, such as product temperature, into commercial refrigeration modeling using EnergyPlus.

3.2.1 Background

While EnergyPlus is a comprehensive whole-building energy simulation program, detailed simulation of certain building subsystems is outside of its native capabilities. One example of this is a refrigerated case. Although EnergyPlus does contain a refrigerated case model, it assumes a constant refrigerated case temperature. In most instances, this assumption is adequate; however, in this example, the dynamics of the refrigerated case (including the product inside the case) are of interest to the user. This application of the Python plugin seeks to enhance a native EnergyPlus capability.

3.2.2 Approach

Here, we used a gray box model written in Python that simulates refrigerated case temperature and product temperature. We used the EnergyPlus Python plugin to connect this gray box model to the native EnergyPlus refrigerated case. Section 18.2.3 of the EnergyPlus Engineering Reference (DOE 2023b) details how commercial refrigeration systems are modeled in EnergyPlus. In this instance, the gray box model is being used to determine two things: (1) the

temperature of the product inside the refrigerated case, and (2) the sensible and latent contribution of the refrigerated case to the thermal zone inside the EnergyPlus model.

Product Temperature

EnergyPlus does not encompass modeling of temperatures of the product inside the refrigerated case. Therefore, the gray box model simulates alongside the EnergyPlus model and extracts information from the EnergyPlus model (e.g., zone temperature) while running. The case credit gray box model described in the next section uses the product temperature as an input, so the product temperature affects upstream EnergyPlus calculations (e.g., the load on the HVAC system), as well as the zone conditions inside the EnergyPlus model.

Case Credits

"Case credit" is the term used for removal of sensible and latent energy from the surrounding environment (typically the EnergyPlus zone) by the refrigerated case. EnergyPlus currently includes calculations for sensible and latent case credits.

Sensible Case Credits

First, a rated sensible case credit value is calculated using fixed case characteristics, such as the latent heat ratio of the case and the installed lighting power (see Equation 18.144 in the EnergyPlus Engineering Reference [DOE 2023b]). Normally, the simulation program (EnergyPlus) would calculate the sensible case credit per time step for each refrigerated case using Equation 18.145 in the EnergyPlus Engineering Reference (DOE 2023b). In this instance, we use the Python model described above to determine the sensible case credit per time step using (in part) the zone temperature and humidity ratio at that time step. In other words, the Python model provides the left-hand side of Equation 18.145, also shown below (DOE 2023b):

$$\dot{Q}_{CC_{sens}} = \dot{Q}_{CC_{sens,rated}} \left(\frac{T_{db,air} - T_{case}}{T_{db,rated} - T_{case}} \right) (SCH_{CC})$$

For implementation in EnergyPlus, Equation 18.145 is solved for the sensible case credit schedule value (SCH_{CC}) and then set a Python API actuator assigned to this schedule. In the model itself we use a placeholder schedule that always has a value of 1, which we then overwrite at each time step with the result from solving Equation 18.145. Finally, the zone conditions are updated for the next time step with this sensible load from each refrigerated case.

Latent Case Credits

The latent case credits are calculated in a very similar manner to the sensible case credits. The Python model provides a latent case credit value for every simulation time step using the zone temperature and humidity ratio from that time step. For implementation in EnergyPlus, we solve Equation 18.147 from the EnergyPlus Engineering (DOE 2023b) for the latent case credit schedule value (SCH_{CC}) and then set a Python API actuator assigned to this schedule:

$$\dot{Q}_{inf,lat} = -\dot{Q}_{CC_{lat}} = \dot{Q}_{case,rated}(LHR_{rated})(RTF_{rated})(SCH_{CC})(LatentRatio)L_{case}$$

Finally, the zone conditions are updated for the next time step with this latent load from each refrigerated case.

Figure 4 shows the data exchange between EnergyPlus and Python for the refrigerated case modeling.

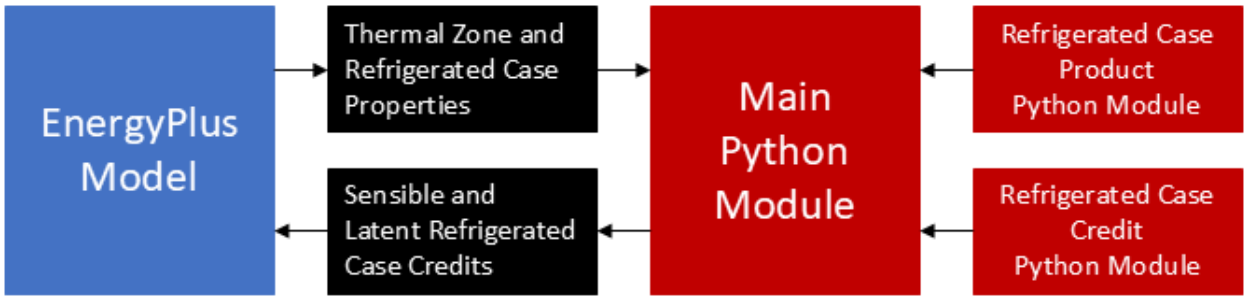

Figure 4. Schematic diagram of refrigerated case Python plugin modeling workflow

Figure 5 shows example EnergyPlus output with the case temperature and product temperature model integrated into a building model via the Python plugin. In Figure 5, the blue line shows the "bulk" or average temperature of the air inside the refrigerated case, and how it changes in response to other model inputs (including refrigeration compressor cycling). Similarly, the orange line in Figure 5 shows the average temperature of the product inside the refrigerated case.

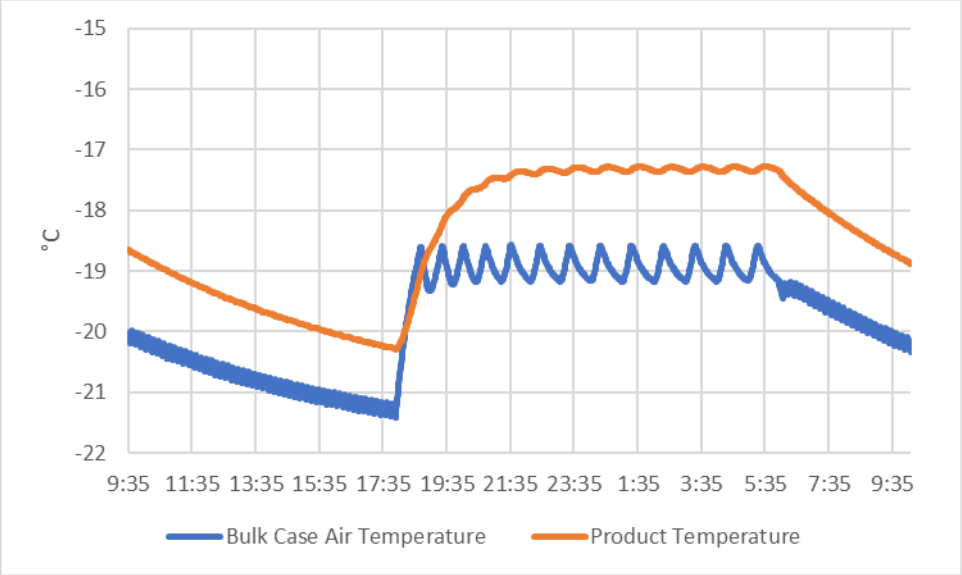

Figure 5. Example EnergyPlus model outputs with Python plugin refrigerated case enhancements

Both the bulk case air temperature and product temperature are outputs that are not native to EnergyPlus, and the Python plugin system allows the user to couple these additional models with the whole-building energy simulation, using inputs (such as zone temperature) from the building model, and modifying the building model's behavior by adjusting the case credits. Traditional EnergyPlus EMS could be used to actuate the case credit schedules; however, co-simulation with the more nuanced Python model that is calculating the case credits is only possible with the new Python plugin.

3.3 Outdoor Air Pretreatment

This case study presents an illustrative example of streamlined modeling of new HVAC technologies within EnergyPlus. The technology is not described in detail, nor are any conclusions presented. Instead, we show how the Python plugin can be used to rapidly create a model using performance data, without the time-consuming process of updating the EnergyPlus code base.

3.3.1 Background

The need to translate tabular equipment performance data into a format that is usable by a simulation program is common in building energy modeling. Typically, a regression approach is used to fit a curve to the data. This preprocessing step is generally cumbersome and time-consuming, with the outcome being a set of regression coefficients that must be accounted for and properly input into the simulation program. Because of this burdensome and error-prone process, generic performance data are often used and deemed good enough. In this example, we illustrate a way that the EnergyPlus Python plugin can be utilized to streamline incorporation of tabular equipment performance data into a building energy model.

3.3.2 Approach

In this instance, we modeled an outdoor air (OA) pretreatment device that conditions OA to a specified dew point temperature. Typically, untreated OA is mixed with return air before entering the intake of an air handling unit, and then this entire mass of air is conditioned by the cooling coils in the air handling unit to meet building cooling needs. Here, the OA is being conditioned before mixing with return air. Treating the OA separately decouples a building's ventilation-driven cooling needs from its internal gain-driven cooling needs. This allows an air handling unit to better match its operation to the building's needs and thus operate in a more energy-efficient manner.

Custom Component Modeling

In this instance, we utilized the scikit-learn machine-learning Python module (Pedregosa et al. 2011). scikit-learn is an open-source, commercially usable, and universally accessible tool for predictive data analysis in Python. With the scikit-learn module, we can, at simulation time:

- Read in performance data for the OA pretreatment device in a comma-separated value (CSV) format.
- Define the independent and dependent variables in the dataset.
- Fit a regression to the dataset.
- Report goodness of fit metrics (e.g., coefficient of restitution and root mean square error).
- Use the regression (at each simulation time step) to simulate the performance of the OA pretreatment device.

Once the regression has been created using scikit-learn, the OA pretreatment device is represented in EnergyPlus with a user-defined component model, specifically the *Coil:UserDefined* object. The *Coil:UserDefined* object is used within EnergyPlus to define a generic coil for custom simulation of a device that processes air as part of an air handling system. This user-defined component model is a shell that provides the user with useful inputs (such as

inlet conditions) and requires certain outlet conditions to be passed from the custom component model to the rest of EnergyPlus. Finally, the Python plugin system:

- Passes the inlet conditions from EnergyPlus to the regression.
- Uses the regression to calculate the OA pretreatment device performance and outlet conditions.
- Passes this information to EnergyPlus to be included in the whole-building energy simulation.

Figure 6 shows the data exchange between EnergyPlus and Python for the OA pretreatment modeling.

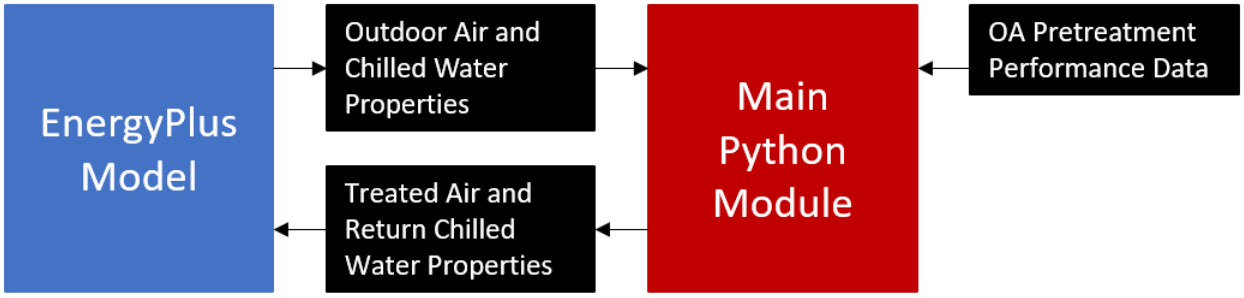

Figure 6. Schematic diagram of OA pretreatment Python plugin modeling workflow

Figure 7 shows example EnergyPlus output with the OA pretreatment model integrated into a building model via the Python plugin. The blue line shows the temperature of the mixed air entering a typical air handling unit, and the orange line shows the temperature of the mixed air entering a typical air handling unit after it has been pretreated.

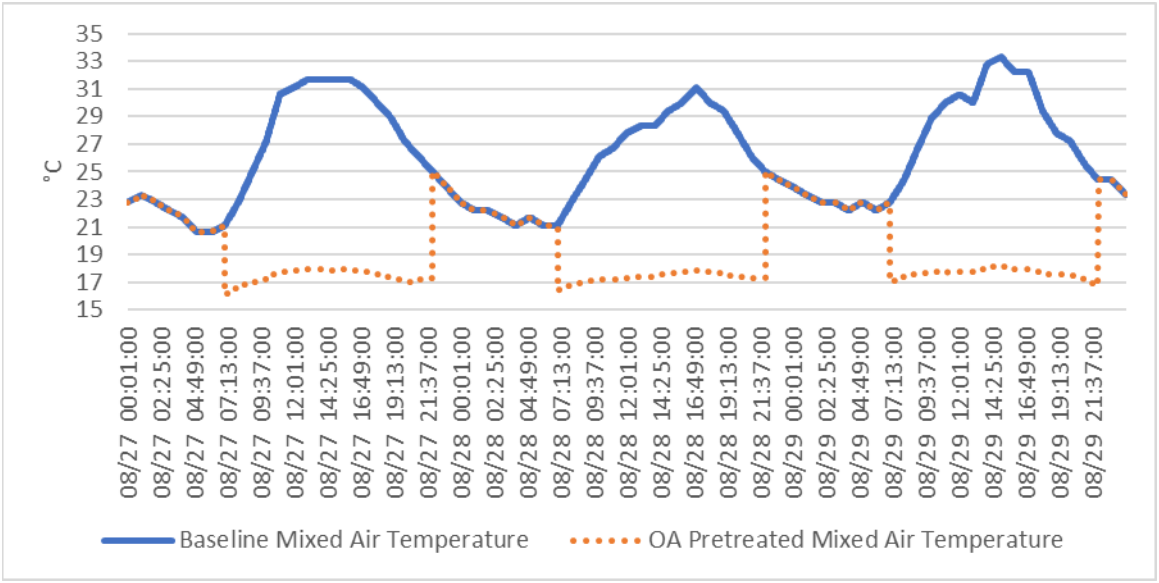

Figure 7. Example EnergyPlus model outputs with Python plugin OA pretreatment device

The orange line is consistently lower during building occupied hours (7:00–22:00), which then allows for the air handling unit to be controlled to a different (higher) leaving air temperature due to dehumidification of the pretreated OA, reducing reheat (and overall building energy

consumption). This type of dynamic coupling of the user-defined component model and the Python-based regression is not possible without the EnergyPlus Python plugin. EnergyPlus uses this Python-based regression model directly, as opposed to articulating a lookup table or curve in EnergyPlus. This not only removes the burden from the user to generate and implement the regression-based model, but also allows for the use of potentially unconventional independent variables. EnergyPlus has a finite, albeit expansive, set of independent variables, whereas the Python plugin would allow for almost any independent variable imaginable.

3.4 Groundwater Heat Exchanger and OpenStudio

This case study presents an illustrative example of the modeling of new HVAC technologies within EnergyPlus. The technology is not described in detail, nor are any conclusions presented. Instead, we show how the Python plugin can be used to create a new model using sophisticated equations, without requiring developers to alter the EnergyPlus code base. This case study is also an example of leveraging the Python plugin through OpenStudio® rather than directly from EnergyPlus.

3.4.1 Background

EnergyPlus offers EMS and ERL for user-defined representation of component behavior that does not exist as objects in EnergyPlus. However, ERL is limited in its ability to represent complex equations. In this example, the Python plugin facilitated use of a complex set of equations to model performance of a novel heat source and sink—a groundwater heat exchanger (GWHE). A study was conducted to characterize the potential performance of a GWHE system coupled with a building HVAC system. In the configuration analyzed, which was based on a system patented and commercially available in the United States, groundwater flow is induced by a circulation pump through a heat exchanger submersed in a wellbore. Water from a building circulates down into the wellbore through the other side of the heat exchanger. With the indirect heat exchanger configuration, there is no consumptive use of groundwater, and no risk of cross-contamination between the working fluid and groundwater. In this configuration, the building loop can serve as the condenser loop for water-source heat pumps or a chiller, or it can be used directly in medium-temperature cooling applications, such as for chilled beams. The study aimed to understand key design parameters of the systems and their influence on energy performance.

3.4.2 Approach

The commercially available technology is relatively nascent, and experimental system performance data were not available in sufficient quantities to develop an empirical model. Additionally, there are not any native EnergyPlus objects within the codebase to model this type of technology. Instead, the performance of the GWHE was characterized by a set of analytical equations. The equations required to characterize the performance of the heat exchanger could not have been represented directly in the ERL used by EMS, given their use of exponentiation and other complex functions. The use of external lookup tables to evaluate these functions would have been cumbersome and error prone. Typical OpenStudio measures, which modify OpenStudio models programmatically through Ruby code, are executed once, and not at each simulation time step, as required for this application. Thus, for this study, OpenStudio measures were used for configuration of the underlying model and to initiate the Python plugin workflow, and the Python plugin was used to model the ground heat exchanger performance.

Use of the Python plugin allowed the analytical equations to be programmed and solved in Python and still interact with the EnergyPlus model at each time step. The analytical equations represent the finite annular flow solution for heat exchange in a tube/annulus pair (Chanson 2004; Harmen and Abdurrachim 2018). This approach also made the workflow modular and readily adaptable to any modeling refinements, as the Python module was separate from the building energy model, generating boundary conditions for the model.

Groundwater Heat Exchanger and Building Model Connection

The Python EMS feature in EnergyPlus is used to connect a foundational Python script (the "root" module) with a building energy model. This implementation includes three Python modules; however, the fundamental requirement for this workflow is a Python module connected to EnergyPlus through the Python API. In addition to the root module, separate Python modules evaluate the heat exchanger performance at each time step (GWHE module) and pump power (pump power module). Figure 8 shows an overview of the architecture of the workflow. This approach corresponds to the use of entry Point A in Figure 1.

The root Python script takes inputs for the GWHE design parameters and loop set points and initiates the calling of helper modules. The root Python script calls the GWHE module to calculate outputs of the GWHE and the pump power module to calculate the ground loop pump power. The root Python script was called at the "after predictor after HVAC managers" EMS calling point at each time step of the simulation. The GWHE module calculates a return temperature to the building condenser loop based on a mass flow rate and entering temperature, using equations describing the heat exchanger's performance. The interactions between the EnergyPlus model and the Python modules are shown in the schematic diagram in Figure 8.

The following steps take place in the Python workflow:

- In the root Python module, condenser water temperature and mass flow rate are read in from EMS sensor objects. This corresponds to Point 2 in Figure 1.
- If a nonzero mass flow rate is present, the condenser loop temperature value is compared to the set point range.
- If the condenser loop temperature value is outside of the range, the mass flow rate and temperature are passed to the Python GWHE module.
- The GWHE module calculates the return water temperature, based on the heat exchanger's effectiveness at the given conditions, and calls the pump power module to calculate the ground loop pump power.
- The GWHE module returns the outlet temperature and pump power to the root module.
- The root module uses EMS actuators for the *PlantComponent:TemperatureSource* and schedule for the *OtherEquipment* object representing the ground loop pump to set the return temperature and pump power, respectively. This corresponds to Point 3 in Figure 1.

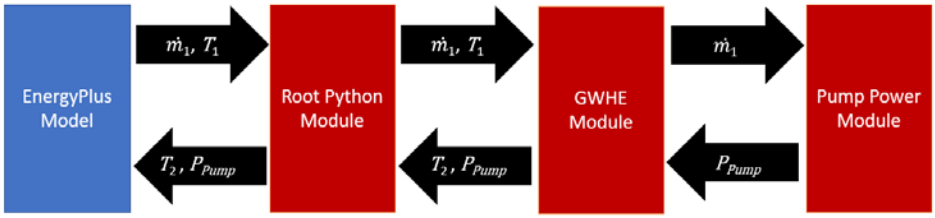

Figure 8. Schematic diagram of GWHE Python plugin modeling workflow

OpenStudio Modeling Workflow

An OpenStudio model of a building with a hydronic HVAC system was modified through custom OpenStudio measures to add an additional hydronic loop (termed the "wellfield loop") that modeled the connection with the GWHE and an *OtherEquipment* object to track the energy use of the circulation pump serving the GWHE, referred to as the "ground loop pump." A modified version of the "Office HVAC WSHP DOAS" measure from the Advanced Energy Design Guide OpenStudio gem was used to create the desired HVAC system, and a custom measure was used to add the required Python EMS objects (NREL 2022a).

The wellfield loop incorporates a *PlantComponent:TemperatureSource* object on the primary side of the loop. In EnergyPlus, such an object can represent an infinite heat source or sink at a given temperature, which can be set by a schedule. An actuator in EnergyPlus' EMS was used to control the temperature to represent the return temperature from the GWHE. The ground loop pump was represented with an *OtherEquipment* object, with its power draw set by a schedule. An EMS actuator tied to the pump's schedule was used to set the power consumption of the pump based on the value calculated by the pump power module. Using OpenStudio measures, this workflow is generally applicable to other commercial or residential building models that are compatible with a hydronic HVAC system.

As an illustration of the workflow's functionality, Figure 9 shows an example of how the ground loop pump is actuated to maintain the wellfield loop temperature within the desired range over the course of a day during the 1-year period analyzed. The ground loop pump effectively maintains the wellfield loop temperature within the desired bounds of 8°C to 17°C. This set point range was selected to establish an appropriate trade-off between heat pump and circulation pump energy. It is important to note that the *OtherEquipment* object is a zone object, and the EMS calling point occurs after its value is calculated. Therefore, a post-processing adjustment is made to account for this temporal lag.

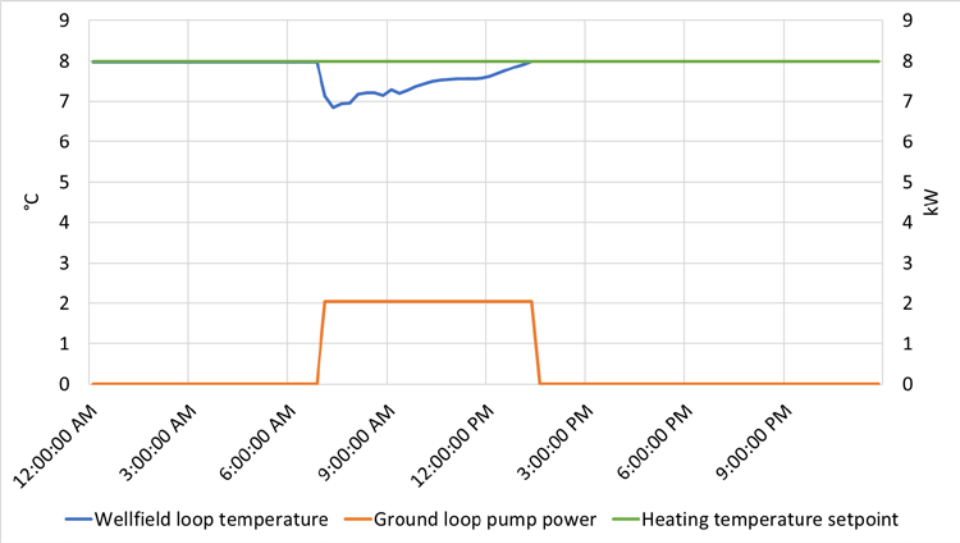

Figure 9. An illustration of how the ground loop pump is actuated to maintain the loop temperature within the desired range. Note that the upper loop set point temperature is 17°C.

The modularity of the Python plugin approach facilitates refinements to the model. One such refinement made by the authors during this study is implementation of a variable-speed ground loop pump in the model. The heat exchanger effectiveness, and thus the return temperature, is dependent on the ground loop mass flow rate. The intention in controlling the variable-speed pump was to reduce the pump flow to the minimum level necessary to ensure that the return water temperature was within the desired range. To avoid having to set up a closed form of the equations characterizing the heat exchanger performance, an iterative approach was implemented in the root Python module to determine the minimum pump speed that would meet the water loop set point temperature.

4 Conclusion

The EnergyPlus Python API and Python plugin provide users with more flexibility, including the ability to call EnergyPlus from Python as a library and to use the plugin to call Python scripts from EnergyPlus. The Python API and plugin help address some limitations of energy simulation tools by allowing advanced control strategies and novel HVAC systems and components to be modeled as part of a whole-building energy simulation. The case studies presented illustrate applications of the EnergyPlus Python plugin to extend and enhance EnergyPlus capabilities and facilitate the integration of performance data for simulation at runtime. The Python plugin allows EnergyPlus to be used in a more flexible manner and to accommodate the expanding realm of energy modeling applications, including hardware-in-the-loop studies, modeling of grid-interactive systems, and modeling of novel heat sources and sinks that can play a role in energy efficiency.

References

Bernal, W., M. Behl, T. Nghiem, and R. Mangharam. 2012. "MLE+: A Tool for Integrated Design and Deployment of Energy Efficient Building Controls." 4th ACM Workshop on Embedded Sensing Systems for Energy-Efficiency in Buildings (BuildSys '12). Toronto, Canada.

Building Systems Laboratory. 1999. *BLAST 3.0 User's Manual*. Urbana-Champaign, IL: University of Illinois Department of Mechanical and Industrial Engineering.

Chanson, H. 2004. *Hydraulics of Open Channel Flow: An Introduction Basic Principles, Sediment Motion, Hydraulic Modelling, Design of Hydraulic Structures*. Elsevier. doi.org/10.1016/B978-0-7506-5978-9.X5000-4.

Crawley, D. B., L.K. Lawrie, F.C. Winkelmann, W.F. Buhl, Y.J. Huang, C.O. Pedersen, R.K. Strand, R.J. Liesen, D.E. Fisher, M.J. Witte, and J. Glazer. 2001. "EnergyPlus: Creating a new generation building energy simulation program." *Energy and Buildings* 33(4): 319–331. doi.org/10.1016/S0378-7788(00)00114-6.

Ellis, P. G., P. A. Torcellini, and D. B. Crawley. 2007. "Simulation of Energy Management Systems in EnergyPlus." *Proceedings of Building Simulation 2007*: 1346–1353.

Gilleran, Madeline, Eric Bonnema, Jason Woods, Partha Mishra, Ian Doebber, Chad Hunter, Matt Mitchell, and Margaret Mann. 2021. "Impact of electric vehicle charging on the power demand of retail buildings." *Advances in Applied Energy* 4: 100062. dx.doi.org/10.1016/j.adapen.2021.100062.

Goia, F., G. Chaudhary, and S. Fantucci. 2018. "Modelling and experimental validation of an algorithm for simulation of hysteresis effects in phase change materials and building components." *Energy and Buildings* 176 (1): 54–67. doi.org/10.1016/j.enbuild.2018.06.001.

Griffith, B. 2023. Personal communication between Matt Mitchell and Brent Griffith. Feb. 21, 2023.

Harmen, W.A., and A.D.P. Abdurrachim. 2018. "Theoretical investigation of heat transfer correlations for supercritical organic fluids." *AIP Conference Proceedings* 1984 020011.

International Energy Agency. 2020. *Global EV Outlook 2020*. www.iea.org/reports/global-ev-outlook-2020.

Lawrie, L. 2009. "EnergyPlus V4.0 Released." Oct. 12, 2009. energy-models.com/forum/energyplus-v40-released.

Mishra, Partha, Jun Myungsoo, Anya Peterson, Eric Bonnema, and Lauren Klun. 2022. *PyChargeModel (Oriented Programming Based Electric Vehicle and Electric Vehicle Supply Equipment Charging Model in Python)*. Computer software. U.S. Department of Energy. doi.org/10.11578/dc.20220824.4.

Muratori, Matteo, Emma Elgqvist, Dylan Cutler, Joshua Eichman, Shawn Salisbury, Zachary Fuller, and John Smart. 2019. "Technology solutions to mitigate electricity cost for electric vehicle DC fast charging." *Applied Energy* 242: 415–23.

National Renewable Energy Laboratory (NREL). 2022a. "OpenStudio Advanced Energy Design Guide Gem." github.com/NREL/openstudio-aedg-gem.

———. 2022b. "Python Opens Up New Applications for EnergyPlus Building Energy Simulation." News story, Jan. 5, 2022. www.nrel.gov/news/features/2022/python-opens-up-new-applications-for-energyplus-building-energy-simulation.html.

Pedregosa, F., G. Varoquaux, A. Gramfort, and V. Michel. 2011. "Scikit-learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–2830. jmlr.csail.mit.edu/papers/volume12/pedregosa11a/pedregosa11a.pdf.

Sardoueinasab, Z., P. Yin, and D. O'Neal. 2018. "Energy modeling and analysis of inherit air leakage from parallel fan-powered terminal air units in EMS in EnergyPlus." *Energy and Buildings* 176 (1): 109–119. doi.org/10.1016/j.enbuild.2018.07.019.

Stripp, Sebastian Fischer, Michela Turrin, and Regina Bokel. 2024. "An Open Computational Workflow, Using the EnergyPlus Python API, for Proactive Building System Control Using Deep Reinforcement Learning and Weather Forecasts to Save Energy." Unreviewed preprint available via *SSRN*. dx.doi.org/10.2139/ssrn.4773992.

Taveres-Cachat, E., F. Favoino, R. Loonen, and F. Goia. 2021. "Ten questions concerning co-simulation for performance prediction of advanced building envelopes." *Buildings and Environment* 191. doi.org/10.1016/j.buildenv.2020.107570.

University of California. 2023. "Ptolemy II." Accessed Feb. 27, 2023. ptolemy.berkeley.edu/ptolemyII/index.htm.

U.S. Department of Energy (DOE). 2023a. "EnergyPlus™ Version 23.1.0 Documentation: Application Guide for EMS." *EnergyPlus Version 23.1.0*.

———. 2023b. "Engineering Reference." *EnergyPlus Version 23.1.0*.

Wetter, M. 2011. "Co-simulation of building energy and control systems with the Building Controls Virtual Test Bed." *Journal of Building Performance Simulation* 4: 185–203.

Wetter, M., K. Benne, A. Gautier, T. Nouidui, A. Ramle, A. Roth, H. Tummescheit, S. Mentzer, and C. Winther. 2022. "LIFTING THE GARAGE DOOR ON SPAWN, AN OPEN-SOURCE BEMCONTROLS ENGINE." Berkeley, CA: Lawrence Berkeley National Laboratory. escholarship.org/uc/item/9c28b4qp.

Winkelmann, F.C., B.E. Birdsall, W.F. Buhl, K.L. Ellington, A.E. Erdem, J.J. Hirsch, and S. Gates. 1993. *DOE-2 Supplement, Version 2.1E, LBL-34947*. Berkeley, CA: Lawrence Berkeley National Laboratory, National Technical Information Service.