AI Based Residential Microgrid EMS System

Funded by EPRI GridEd Program

Tim Mannon, Jordan Suggs, Harley Dukes

March 2021

Table of Contents

1 Residential Microgrid Simulation Software

The Residential Microgrid Simulation Software is a set of simulators and programs that allow for the electrical energy flow modeling of a distributed generation and storage-enabled residentialmicrogrid. For this project, the microgrid is considered to have a solar generation array and lithium ion battery storage. The program consists of a Non-AI Residential Energy Simulator that creates the "base case" for all scenarios to be modeled and a Residential Microgrid AI EMS Systemwhich incorporates a Grid AI EMS and Residence AI EMS. A scenario could be a different characterarrangement in a home, or a different number of households connected to the same microgrid.The goal of the system is to maximize the energy used from the solar generation and batterystorage and minimize the energy used the main/utility-scale grid. This approach has benefits tothe customer in the form of energy cost reduction and increase in comfort, as well as benefits to the grid operator in forms of preservation of grid components. Another critical beneficiary of such a system is the earth, as the shift towards decreased energy used from the grid translatesto less CO2 emissions.

This section details the working components of the Residential Microgrid Simulation Software.The four scenarios considered consist of a residence with a single occupant who works a normal 40-hour work week, a retired couple, a family of four with grade school-aged children, and a scenario where all three of the previous households are all connected to the same grid. This software is written in Python and incorporates the following libraries/APIs for machine learning(ML):

- TensorFlow (backend for supervised and reinforcement learning libraries/APIs)
- Sci-kit Learn (backend library for unsupervised learning elements)
- Keras (API for supervised learning artificial neural networks – TensorFlow backend)
- Stable Baselines-RL (API used for reinforcement learning agent creation and implementation – TensorFlow backend)
- OpenAI-Gym (API for creating environments for reinforcement learning agents – integrates with Stable Baselines-RL).

Using these libraries and APIs greatly reduced program development time and allowed for rapid prototyping.

2 Non-AI Residential Energy Simulator

Each model is originally simulated using a residential simulator that has no machine learning (ML)or AI components (NonAI_Sim_1_0_(Scenario#).py). This simulator simply creates the charactersand records their behavior for one year. The simulator also models the energy usage for the givencharacter behavior assuming no AI EMS, solar generation/battery storage, or dynamic TOU pricing. This one-year simulator runs for 364 days or "episodes" with 96 increments or "time steps" for each episode. This translates to 15 simulated minutes per time step, with each episode representing a normal 24-hour day. There are 34944 timesteps for each year. For simplicity in

time array generation and seasonal considerations, the simulation year starts on December 21and ends on December 20.

The simulator creates the "base case" for each character scenario. The behaviors of the characters in this base case are used when simulating the AI-enabled residence in order for fair comparison of final results. All ML elements are trained based on the output from the initial base case created in the non-AI residential simulator. More detailed descriptions of key simulator components are given in the next sections.

2.1 Time Vector Creation

To ensure consistency between models and for efficient training of ML elements, a time stampfor each increment of the simulation must be created. A function was created to construct a timestamp for each increment in vector form. The result is an 87-element binary vector, which is divided up by length in the following four sections: hour of day (24), increment of hour (4), dayof week (7), week of year (52). The program also converts this binary vector into a 4-elementscalar vector. Both are used for ML element training and timing for the AI Residential EMS.

2.2 Character Behavior Array Creation

The ability to generate characters that behave in a somewhat random way was critical to success of this project. While behavior has aspects of randomness, there needs to be some consistencyor pattern in behavior, as is the case in real-world day-to-day human behavior. Another important consideration is to model the character behavior only in a way that is seen from theperspective of the Residential AI EMS system. With these considerations, the Character BehaviorArray (CBA) consists of five columns for each timestep: home or not home indication (1 = home,0 = not home), temperature preference (scalar – degrees Fahrenheit), shower state (1 = showerin use, 0 = shower not in use), dishwasher state (1 = dishwasher on, 0 = dishwasher off), and laundry state (1 = laundry machine on, 0 = laundry machine off). Using these 5 indicators, theenergy use for the four appliances can be modeled and/or controlled.

CBAs were created for three different household scenarios: 1 – single adult who works a regular 9:00AM – 5:00PM Monday-Friday workweek who is free on the weekends, 2 – retired couple that has no regular work schedule but similar seasonal patterns between simulation years, 3 – familyof four with parents that work a regular work schedule with children in school. For each scenario behavior of the household residents was modeled in such a way that would be seen from theperspective of the Residential AI EMS. For simplicity, It was assumed that behavior was consistenton a week-to-week basis, meaning there were no major anomalous incidents such as an extendedvacation or "sick days". While in the real world these anomalies are present, it is assumed that a user-setting would exist that allowed residents to indicate when these periods are occurring. For the sake of this proof-of-concept project, it was decided that leaving out anomalies would be the best approach. In future models, the use on anomalies in character simulation will be explored.

For home or not home output, the leaving and arrival times of the residents were handledpseudo-randomly for Scenario 1, completely randomly for Scenario 2, and a combined approach was applied for Scenario 3. In Scenario 1, the weekday departure and arrival times were selected randomly each day but would fall within certain ranges. Departure times were between 7:00 –8:45 AM and arrival between 4:00 – 6:45 PM on weeknights. On weekends, the departure andarrival times were completely randomized, by first giving a random time for first departurebetween 6:00 AM and 6:00 PM. From there the number of departures was randomized based on the time remaining in a day from the first departure time. This allowed from a range of departuresand arrivals in a given day between 1 and 10 times. The character was always assumed to have arrived home before 11:45 PM each day on weekends.

The home or not home array for Scenario 2 was constructed using the weekend approach inScenario 1, but this was applied to everyday of the week for Scenario 2. Between years, the year 0 array was shuffled using a sliding window shuffler of 7-day length. This means the daily arrival- departure patterns initially created in the first year were preserved but shuffled to a different day in the 364-day simulation in later years. Using a sliding window of 7-days, ensure that similar coming and going patterns were maintained at similar times of the year as in the initial simulation. That was done to preserve the seasonal behavior between years that most households present, instead of completely random behavior.

Scenario 3 used a combination of approaches from Scenarios 1 and 2. Because the householdcontained two characters that were school children, arrival and departure behavior needed tobe different between school year and summer break times. Summer break was assumed to start on May 15 and end on August 15. During the school year, weekday departures and arrivals were applied similar to Scenario 1 with larger windows. Departure times could range between 6:00and 9:45 AM and arrivals from 2:00 and 9:45PM. This was done to encapsulate the various extra-curricular that may exist for the children and also the fact that school children are often homebefore their parents are finished with work. During the school year, the weekends are handledin the same way as Scenario 1. During summer break, the arrival and departure behavior is created in the same manner as Scenario 2. This was done to simulate the random behavior/schedule of school children during summer break. The year-to-year behavior is not handled the same as Scenario 2, and completely new arrival and departure times are generated during summer break between years for Scenario 3.

The temperature preference arrays were constructed to account for differences in temperaturepreference throughout the day and also seasonally. The daily temperature preference changes were made to account for temperature when individuals are waking up in the morning, throughthe middle of the day, and when they go to bed, leading to three different temperature preferences throughout the day. The time at which the temperature preference would changebetween day sections was randomized each day within a range. Tables 1 – 3 show the temperature preference for each Scenario for 1 year.

Table 1: Annual Temperature Preferences for Scenario 1

Season	Wake Up Temp (° F)	Mid-Day Temp (° F)	Sleep Temp (° F)
Winter	73	72	70
Spring	72	70	69
Summer	70	70	69
Fall	72	71	70

Table 2: Annual Temperature Preferences for Scenario 2

Season	Wake Up Temp (° F)	Mid-Day Temp (° F)	Sleep Temp (° F)
Winter	74	72	72
Spring	73	71	70
Summer	72	72	70
Fall	73	71	70

Table 3: Annual Temperature Preferences for Scenario 3

Season	Wake Up Temp (° F)	Mid-Day Temp (° F)	Sleep Temp (° F)
Winter	73	72	70
Spring	72	71	69
Summer	72	70	69
Fall	72	71	70

The shower, dishwasher, and laundry state arrays were randomized based on the arrival departure behavior for each scenario. When scenarios exhibited the normal weekday work schedule, showers were always initiated at a random time before the first departure time.Shower duration was modeled as 2 timesteps (or 30 minutes). For multi-character households,the number of showers taken each day was randomized with 1 shower a day minimum up to the total number of individuals in the household being the maximum number of showers each day.For non-regular weekday work schedules, showers can occur at any time throughout the day.

The dishwasher and laundry times were handled similar to showers. Dishwasher run times were modeled at 6 timesteps (90 minutes) and laundry at 8 timesteps (2 hours). It was assumed thatthe households had an all-in-one washer and dryer laundry machine. Dishwasher and laundry runtimes were initiated at random times after the latest arrival times for scenarios that consisted ofregular weekday workweek patterns. On the weekends, and on any day for Scenario 2 and Scenario 3 – Summer, dish and laundry could be initiated at any time. The number of times dish and laundry machines were ran in a given week was randomized based on the number of characters in each household. The random number of times each appliance was ran for eachscenario fell within the following ranges: Scenario 1 – dishwasher(2 – 6), laundry (0 – 4); Scenario2 - dishwasher(3 – 7), laundry (2 – 7); and Scenario 3 - dishwasher(2 – 7), laundry (2 – 7).

2.3 Import Weather and TOU Pricing

Weather data for Chattanooga from December 2017 to December 2018 was used to model the outdoor temperature. Historical weather data for Chattanooga was found at: https://api.wunderground.com/history/airport/KCHA/1936/1/1/DailyHistory.html?&reqdb.zip=&reqdb.magic=&reqdb.wmo=
(WeatherDataInterpolated_10-19-19.txt) Since only one year of data was collected, the weather between years was shuffled using the sliding window shuffling approach as explained before for Scenario 2 departure and arrival behavior. This approach was applied in order to conserve seasonal temperature behavior but present different temperatures each day between years.

The TOU pricing guide was based on San Diego Gas and Electric utility TOU pricing in 2019 (TOUData4Import_10-19-19.txt). This TOU pricing contains only a base price and peak pricing perkWh. Peak times are considered to be between 4:00 and 9:00 PM every day. This cost array was used to determine the base case annual energy cost for each scenario. Both the weather dataand TOU pricing data imports are 34944 elements in length, ensuring a value for each timestepin a 1 year simulation

2.4 Energy Use Simulation

Using the CBA, imported weather data, and imported TOU array, the energy usage and costingfor each scenario was modeled. The simulator models the energy used by the dishwasher, laundry machine, hot water heater, HVAC, and general energy usage based on the inputs fromthe CBA. The energy used by each element is totaled each timestep and multiplied by the TOU cost per kWh for that timestep. At the end of the 364-day simulation, energy usage and cost foreach timestep are totaled to reveal the final annual energy usage and cost. The water and electrical power usage data for each element are given in Table 4.

Table 4: Hot Water and Electrical Energy Usage Figures for Each Element Per Timestep

Element	Hot Water Usage (gal/15min)	Energy Usage (kWh/15min)
Shower	5	0
Dishwasher	0.75	0.2055
Laundry	0.5	2.06
Hot Water Heater	3.75	1.14
HVAC	0	0.81
General Energy Use	0	0.15 (Away) 0.3 (Home)

For shower, dishwasher and laundry, usages are only applied when a 1 value exists for thosedevices in CBA. The Hot Water Heater is turned on when the hot water level falls below a certain threshold. The hot water tank is assumed to be 80 gallons, and the threshold was set at 65 gallons. Water level is calculated based on the usages from shower, dishwasher, and laundry.General Energy Use is applied based on whether a character is home if the house is empty. Whenno characters are home, the base rate of 0.15 kWh/timestep is applied. When a character is homea general energy use of 0.3 kWh/timestep is applied. For HVAC, the system is on when the

internal temperature of the household falls outside of 1 degree from the temperature preferencefor that timestep based on the CBA. It is assumed that the HVAC can automatically switch from heating and cooling, and there is no limit to how many times it switches from heating to coolingfor any given day.

2.5 Saving and Reporting

After the energy simulation is ran, the program outputs many key arrays that are used for training ML elements and/or modeling behavior in the AI-based simulations. A list of key output arrays is given below:

- Full Character Behavior Array
- Input vectors for temperature behavior neural network
- Input vectors for temperature preference neural network
- Input vectors for home/not home prediction neural network
- Energy usage for each timestep
- Dish state array
- Laundry state array
- Shower state array
- Home/not home state array
- Temperature preference array
- Time vector (binary and scalar)

In addition to saving key arrays in .txt format, the simulator also outputs a display of performance results that are used for model comparisons (Figure 1).

```
HVAC Percent = 17.415838929638262
HWH Percent = 14.855598174747072
Dish Percent = 1.828397026599392
Laundry Percent = 24.46760011264393
General Percent = 41.43256575637132
Annual kWh Used = 20972.633100000003
Annual kWh Cost = 5463.924946308
Total Number of Timesteps using HVAC: 4560
Timesteps when temp outside of 2 degrees from preference: 2526
Total Number of Timesteps using HwH: 2971
```

Figure 1: Example of Scenario 3 Output Display for Non-AL Residential Simulator

3 Residential Microgrid AI EMS System Overview

(ResAI_GridEMS_RT_FullModel_Main_SingleHome.py,
ResAI_GridEMS_RT_FullModel_Main_ThreeHome.py)
The full Residential Microgrid AI EMS System includes a Residential AI EMS and a Grid AI EMS thatexchange minimal information in order to maximize energy used from solar generation and/orbattery storage and minimize energy used from the main/utility-scale grid, which is likely sourced from fossil fuel generation.

Figure 2 shows how the two AI EMS systems interact for a 1-homescenario.

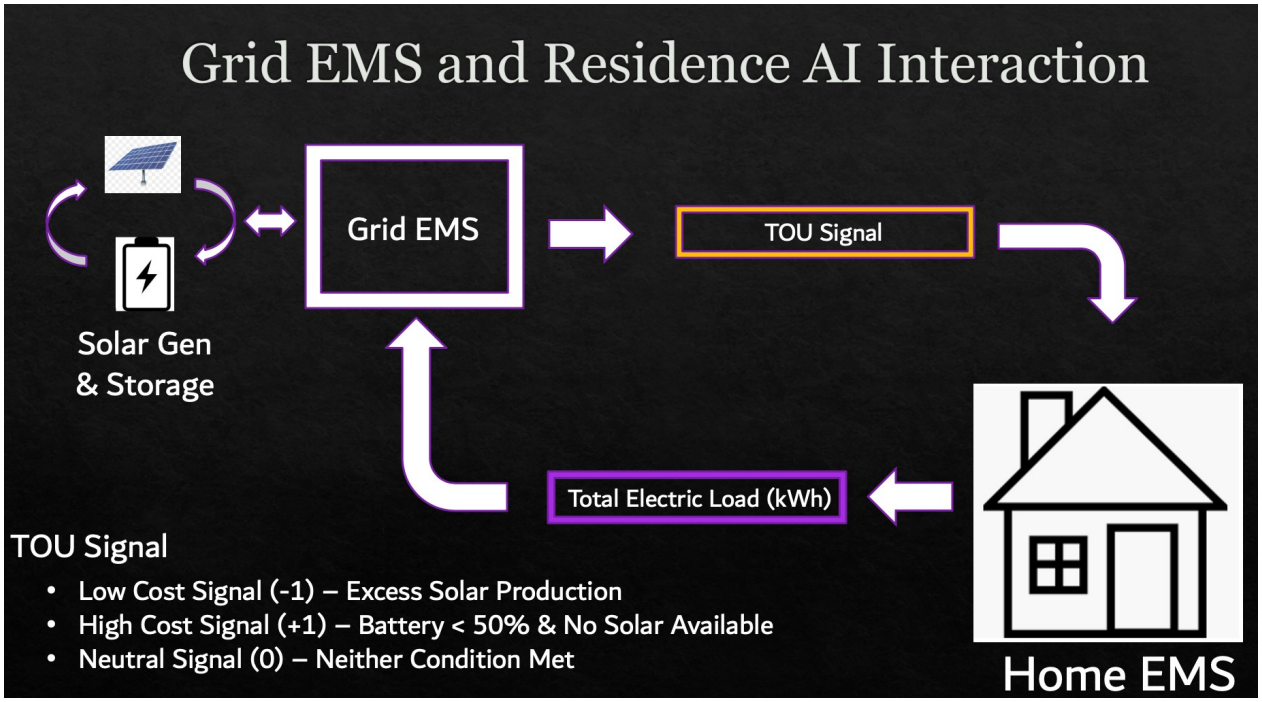

Figure 2: Grid AI EMS and Residential AI EMS Interaction

As is the case with most utilities, the grid EMS only receives the total electric load from theresidence. For simplicity this is converted to kWh in the simulation. The Grid AI EMS creates adynamic TOU signal based on predicted load and simulated solar generation and battery storage.This TOU signal has three settings as shown in Figure 2. A low-cost signal (-1) is sent when thereis expected to be solar generation in excess to what can be used by the residential load or tocharge the grid battery. A high-cost signal (+1) is created when the predicted battery storage levelis less than 50% and no solar generation is projected for that given timestep. The 50% was set in accordance with utility-scale lithium-ion battery storage operational guidelines, that state to limitdropping the charge state of the batteries below 50% as much as possible to conserve the battery lifespan. A neutral signal (0) is sent when neither of the previous two conditions are met for agiven timestep.

A more detailed layout of how the Grid AI EMS and Residential AI EMS function is displayed in Figure 3.

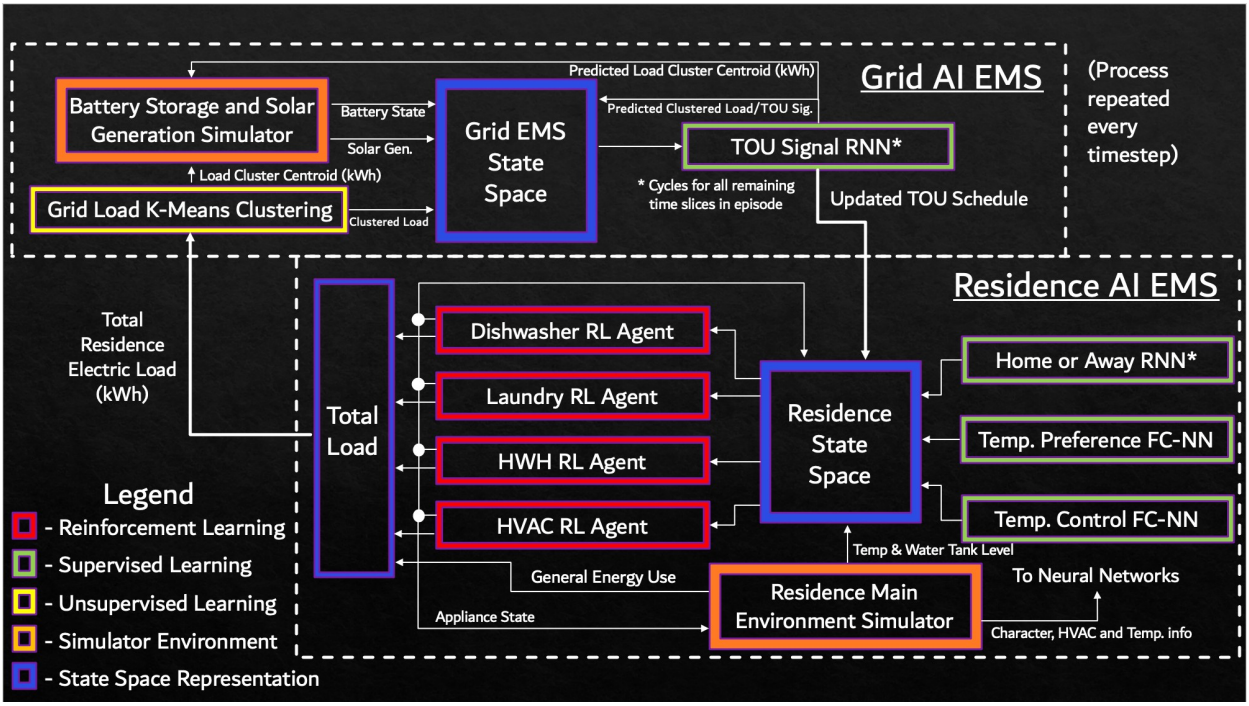

Figure 3: Residential Microgrid AI EMS System Block Diagram

The full Residential Microgrid AI EMS System incorporates 9 separate ML elements from all three major ML categories (supervised, unsupervised, and reinforcement learning). The process shown in Figure 3 is repeated each timestep (15 minutes simulation time), which means the process is carried out 96 times each simulation day or episode.

It is important to note that the TOU Signal RNN and the feedback loop within the Grid AI EMS is carried out for all remaining timesteps within an episode/day. This means that if the simulationis on the first timestep of an episode, the TOU Signal RNN and ensuing feedback loop will cycle 95 times to create a projected TOU Signal Schedule, which is what is sent to the Residence AIEMS. This projected TOU Signal Schedule gives the Residence AI EMS the TOU plan for theremainder of the day, which allow the system to plan appliance usage accordingly. The Home or Away RNN also cycles for all remaining timesteps in an episode to give a predicted home or awayschedule for the remainder of the day. All of the elements in Figure 3 will be discussed in moredetail in the next sections.

The system in Figures 2 and 3 show the communication process for a single Residence AI EMS and the Grid AI EMS. For multi-home examples, the system is similar except the total load fromall the homes are summed each timestep before they are read by the Grid AI EMS (Figure 4).

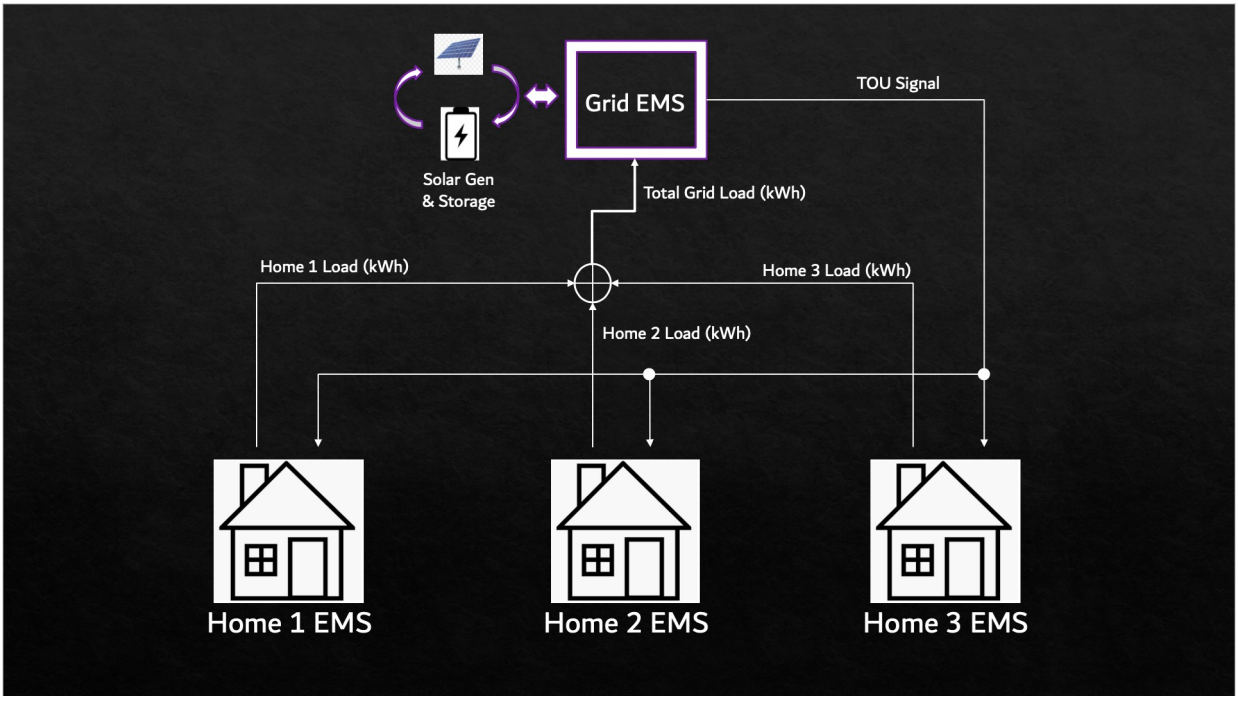

Figure 4: Three-Home Residential Microgrid AI EMS System

In the multi-home scenario, the TOU signal is based on the load forecasts for the combined grid, so the Grid AI EMS does not know the energy usage behaviors of the individual homes. Thismeans that if one of the homes is using energy inefficiently, then all of the homes could see negative effects from the TOU signal, and therefore total energy costs will be higher. This model could be optimized to account for individual home energy forecasting and pricing, but there was no time to implement such a system for this phase in the project. Simulation time would beexceedingly long for such a design, which must be considered in future work for this project.

4 Residential AI EMS

The Residential AI EMS uses many of the aspects to model energy usage as the Non-AI based simulator but incorporates seven ML elements to control when appliances are used and shapethe state spaces for the Reinforcement Learning (RL) Agents that control each appliance. Themajor elements of the Residential AI EMS are listed below:

- Home or Away Recurrent Neural Network (RNN)
- Temperature Preference Fully Connected Neural Network (FC-NN)
- Temperature Control FC-NN
- Dishwasher RL Agent
- Laundry RL Agent
- Hot Water Heater (HWH) RL Agent
- HVAC RL Agent
- Residence Main Environment Simulator

The ML elements in the Residence AI EMS are all trained using the CBAs and other elements fromthe base case or non-AI simulator.

4.1 State Space Support Neural Networks

The neural networks (NN) in the Residence AI EMS used for state space support include the Homeor Away RNN, Temperature Preference FC-NN, and Temperature Control FC-NN. This section explains the data processing, architecture, training, and operation for each NN. Each state space support NN was created using Keras API in Python with TensorFlow backend.

The vectors used in the input class for each NN are taken from the non-AI simulator and are detailed in Table 5. All scalar elements are normalized using Equation 1 as below:

$$x_{norm}{}_i^k = \frac{x_i^k - \mu_i}{\sigma_i}$$

Table 5: Input Class Vector Contents for Each State Space Support NN

Neural Network (Input Class Vector Size)	Input Class Vector Contents (Type)(Size)
Home or Away RNN (88)	• Character home/not home (Binary)(1) • Time Vector (Binary)(87)
Temperature Preference FC-NN (5)	• Temperature Preference (Scalar)(1) • Time Vector (Scalar)(4)
Temperature Control FC-NN (4)	• HVAC On/Off (Binary)(1) • Heating or Cooling (Binary)(1) • Current Indoor Temp. (Scalar)(1) • Current Outdoor Temp. (Scalar)(1)

The Home or Away RNN is intended to take in the previous 24 hours (96 timesteps) of home and not home status and predict the home or not home state for the remainder of the episode. Fortraining purposes, the RNN need only to predict the home or away state for the 97th time slice,or the first time slice after the 96 input time slices. Then in the RT simulation this output can berotated around to the input side and the process repeated until the remainder of the time slicesfor a given episode have been analyzed.

When considering the final use plan for the Home or Away RNN, the processing plan for the input class vectors can be shaped accordingly. The input class vector taken from the non-AI simulation is of shape (34944, 88). Given that the Home or Away RNN will contain 96 LSTM cells, eachtraining case will have 96 timesteps of shape (1, 88). The labeled output for one training example will be the home or away state on the 97th time increment. Using the vector taken from the non-AI simulator, a loop can be initiated that takes a given row and pairs it with the home/not homestate of the next timestep. This will yield a vector of size (34944, 89). Another loop can be initiated

that stacks the examples on top of one another in groups of 96 timesteps, resulting in a three-dimensional vector of shape (34848, 96, 89). This yields a vector consisting of 34848 trainingexamples of 96 timesteps with input class vectors (88) and labeled output (1) to make 89 for athird dimension. This vector is then randomly shuffled along the first axis so there is no order orgrouping for the training set. The labeled outputs are then taken from the main array to yield theinput class training vector (X) and labeled output vector (Y) of shapes (34848, 96, 88) and (34848,96, 1) respectively.

After the input class vector data is processed, the data should be divided into training, validation, and testing categories. The NNs will only be trained using the data selected for training, with validation and testing sets serving as blind tests. For this project the input class vector trainingexamples were divided in the following proportions: training (70%), validation (15%), and test(15%). Randomizing the order of the training examples before making the groupings into training, validation, and test ensures an even distribution of training examples from all points across the1-year simulation. Having a random order of training examples also leads to a more robust NN.

The RNN architecture used for the Home or Away RNN uses 96 consecutive Long-Short TermMemory (LSTM) cells, each with 64 neurons and Sigmoid output activation function (Goodfellow et. al, 2016). For training, binary cross-entropy is used as the loss function with Adam optimization for momentum control during back propagation (Goodfellow et. al, 2016) The Home or Away RNN is trained for 100 epochs. Accuracy results are based on the final LSTM cell output,or 97^{th} timestep, which is close to 100% for all scenarios. During real-time simulation, output ofthe 97^{th} output is taken and cycled back to the input for 96^{th} LSTM cell and paired with the timevector for that representative time slice. The rotation is repeated until the remaining number oftimesteps in the current episode have an estimation of home or away state. For example, if thesimulation is in the first timestep of an episode, the Home or Away RNN is cycled 95 times tocreate a predicted home or away schedule for the remainder of the day.

The Temperature Preference FC-NN is used to estimate the temperature preference within a given household at any given timestep. The input class vector will simply be the scalar time vectoras the training inputs (X) and the temperature preference at each given timestep as the labeledoutput (Y). The input vector with the X and Y parts is of size (34944, 5). This vector is randomlyshuffled along the first axis and separated into inputs and output vectors of size (34944, 4) and(34944, 1). The samples are divided into training, validation, and testing groups as defined inprevious NNs.

The Temperature Preference FC-NN contains 6 fully connected hidden layers that implement the RELU activation function (Goodfellow et. al, 2016). The hidden layer neuron arrangement is (64, 64, 48, 24, 16, 8) with a linear regression output layer, yielding a scalar output (preferredtemperature in degrees Fahrenheit). A mean squared error loss function is implemented with Adam optimization for back propagation (Goodfellow et. al, 2016). This neural network was trained for 500 epochs with a batch size of 16. The resulting accuracy ranges from 98 – 99%between scenarios. During the real-time simulation, the time vectors for a given episode are ran all at once during the first time slice of that episode to generate a schedule for predictedtemperature preference for that day.

The Temperature Control FC-NN was created in order to predict the thermal behavior of the simulated home. The NN takes in the current indoor and outdoor temperatures along with theHVAC controls to predict what the temperature will be in the next timestep. This NN is used bythe HVAC RL to determine what the temperature change will be for a given action, which include: no action (HVAC off), heating, or cooling. The input class vector is processed in a similar manner as the Home or Away RNN, in that the input

for a given training example has a labeled outputthat is the indoor temperature of the next timestep. A loop is implemented to paid the vector fora given timestep (X) with the indoor temperature of the next timestep (Y). This will result in anarray of shape (34943, 5). The array is shuffled along the first axis and separated into input X(34943, 4) and Y (34943, 1). The training examples are then grouped into training, validation, and testing sets.

The architecture, output activation, and back propagation parameters for the Temperature Control FC-NN are the same as the Temperature Preference FC-NN. The NN is trained over 600 epochs with batch size of 16. Accuracy for prediction is 99%+ for all scenarios. During the real-time simulation, this NN is ran each timestep using the current indoor and outdoor temperatureconditions and all three possible HVAC states to yield a prediction for next step temperature given any possible action the HVAC RL agent can take. This information is given to the HVAC RL agent in the observation state space for use in deciding the next HVAC action.

4.2 Appliance RL Agents

RL agents were created to control the Dishwasher, Laundry, Hot Water Heater (HWH) and HVAC appliances. The agents were created in Python using the Stable Baselines-RL API, which uses TensorFlow backend. The RL methodology applied implements a Deep-Q Network (DQN) agent, which is a standard RL agent class within the Stable Baselines-RL API.

Q learning is an off policy temporal difference (TD) reinforcement learning approach with Lambda value 0 (Sutton and Barto, 2018). Q Learning approaches usually implement a non-linear function approximation technique, such as artificial neural networks, to link states to action Q values for policy formulation (Sutton and Barto, 2018). When the non-linear function approximation technique is a deep neural network, the approach is considered Deep Q Network (DQN) learning, which has recently led to technological breakthroughs in literature for solving RL problems in high order state spaces with large action spaces and sparse rewards (Mnih et. al, 2015). The DQN agent implemented for this project incorporates double Q Network (Hasselt et. al, 2015) with dueling agents (Wang et. al, 2015) and prioritized experience replay (Schaul, et. al, 2016). The non-linear function approximator is an FC-NN MLP with two hidden layers each consisting of 40 neurons. This was the only function approximator available for the DQN Agent class within the Stable Baselines-RL API that did not include the use of Convolutional Neural Networks (CNNs), which were not necessary for this project. If a custom DQN agent is to be made in the future, a more robust function approximator would likely be constructed.

In order for an RL agent to function, it must have an environment to step through. This environment contains the state space, reward system, and environmental reset parameters necessary to train RL agents. The Stable Baselines-RL API requires that the custom RL agent environments be integrated with OpenAI-Gym, which is an API created by OpenAI that allows for easy construction for RL agent environments for rapid prototyping. OpenAI does contain several default environments, but none were a good match for this project. Custom environments werecreated for each appliance RL agent except for Laundry, as the Dishwasher and Laundry Agents function in the same way with different input data, so only one environment is needed for both Agents. RL Agent environments were created as Python Classes that integrate the Env object classfrom the OpenAI-Gym API.

In order to integrate an RL agent environment class with OpenAI-Gym, the RL agent environment must contain four key methods: Step, Reset, Render, and Close. The Step method requires the input action and based on this

input assigns rewards based on the previous state and the current action. The state space is then updated and sent back to the agent in this method. The Resetmethod takes in the end of episode indicator as an input, if the end of episode indicator is positivethen this method will reset the state space to reflect the starting state of an episode. The Render method provides the user a way to view the progress of the agent and also can be used for troubleshooting. The Close method is not necessary for all environments and was not used forthe RL agent environments in this project. Figure 5 shows some pseudo code that depicts what a RL agent environment class would require including these methods. Custom RL agentenvironments can be integrated with OpenAI-gym API using the steps detailed at: https://github.com/openai/gym/blob/master/docs/creating-environments.md

```
class SomeAgentEnv(gym.Env):
    metadata = {'render.modes': ['human']}

    def __init__(self):
        self.no_features4state = 71
        self.state = []
        self.episodeEnd = False
        self.starting_state = 0

        # other key parameters such as state and action space dimensions are defined here

    def step(self, action):
        reward = -1
        # here rewards are applied based on the action and the previous state presented to the Agent

        # methods are used to update the state space based on the action taken

        return[self.state, reward, self.episodeEnd]# next state, reward, and episode end indicator are returned

    def reset(self):
        if self.episodeEnd:
            self.state = self.starting_state
        # when episodeOver is True the environment is reset and the starting state of the episode is returned

        return[self.state]

    def render(self, mode='human'):
        # if the render method is used, the user will be able to see the agent's progress through the environment
        print("You Can See What I'm Doing!")
        # variables for troubleshooting or progress tracking can be returned if needed

    def close(self):
        # this is not used in this project
        pass
```

Figure 5: Pseudo Code Example for RL Agent Environment using OpenAI-Gym Standards

The three environments created to train the appliance RL agents were all different due to thedifferent inputs needed for the state space of each agent. The details of each appliance RL agent environment are given in Table 6.

Table 6: Key Parameters for RL Agent Environments

Element	Dish/Laundry Env.	HWH Env.	HVAC Env.
Action Space	Binary[1 = on, 0 = off]	Binary[1 = on, 0 = off]	Discrete [1 = off, 2 = on: cooling, 3 = on: heating]
SS Type	Binary Vector	Scalar Vector	Scalar Vector

State Space Contents	Steps until min cost, Steps until max cost, Appliance state, Appliance enable, Steps until enable expire	Steps until min cost, Steps until max cost, Home or away, Appliance state, Water level, Steps until home	Episode increment, Steps until min cost, Steps until max cost, Steps until home, Home or away, HVAC on, heating/cooling, Current indoor temp, Temp pref. when home, Outdoor temp, Next temp if Action = 1, Next temp if Action = 2, Next tempif Action = 3, Temperature difference from pref., HVAC on frequency in current episode, HVAC use below daily limit
Step Method Contents	• Rewards • Enable Update • Steps until Min/Max TOU Signal Update • End of Episode Ind. • State Space Update	• Rewards • Steps until Min/Max TOU Signal Update • End of Episode Ind. • Steps until home method • Water tank level update • State Space Update	• Rewards • Steps until Min/Max TOU Signal Update • End of Episode Ind. • Steps until home method • Temp. Pref. when home method • Indoor Temp. update • Temp. next step basedon action method • State Space Update
Reward Prioritization	Highest Priorities: Appliance run only when enabled. Appliance run within 20 hours of enable initiation. Lower Priorities: Appliance used during min cost signal and not during max cost	Highest Priorities: Tank not empty when home. Appliance not run if tank full. Lower Priorities: Appliance used during min cost signal and not duringmax cost	Highest Priorities: Temperature kept within +/-1 ° F of preference when someone home. Appliance not used more in one day than daily allotted uses. Lower Priorities: Appliance used during min cost signaland not during max cost

It is important to note that all RL agents incorporated rewards that stayed within +/- 1 in total value. This is to simplify the training within the embedded NN in the DQN agent and preventsvanishing or exploding gradients. For all agents, positive rewards were given when the appliance was used during a low TOU cost signal timestep. The largest rewards in magnitude were negativerewards if the agent took action that caused a violation in the highest priority designations foreach agent. These violations yielded a reward of -1. The highest positive reward used was + 0.75 for all agents. For each environment, the episode length was 96, and

364 episodes were used, which equates to one year of non-AI simulation observation. Each training environment incorporated a shuffle feature in the Reset method, which shuffled the order of simulation inputsat the end of each training year, or every 364 episodes. This creates robustness in the trainingprocess and prevents bias from occurring. When shuffling each year is utilized, the agents aremore adaptable to new scenarios, i.e. a new simulation year.

The Dishwasher/Laundry RL Agent was designed to run the dishwasher/laundry machines during low cost TOU signal time intervals if the appliances have been enabled by a character. If a lowcost TOU signal is not encountered within 20 hours of enable, the agent is trained to run the appliance in order to ensure the appliance is ran within 24 hours of enable. For training, the dish/laundry state vectors from the CBAs are used as the input for enable. These arrays are cleaned to ensure that only a single "1" occurs when the appliance is used, instead of 6 or 8 1s in a row that indicate the duration of when the appliance is on. A simulated TOU signal is createdusing the Solar/Battery Simulator described in later sections based on the energy usage behavior in the non-AI Simulation for each scenario. The simulated TOU signal and Dish/Laundry enable arrays are all that are required for inputs for training the Dish/Laundry RL Agents. These agents are trained for 175,000 timesteps or approximately 5 simulation years.

The HWH RL Agent is designed to heat water during low TOU signal timesteps, but not at thetradeoff of the tank becoming empty while a character is home. Also, the agent is trained not toheat the water if the tank is full. For training, the shower, dishwasher, and laundry state arraysfrom the CBAs in the non-AI simulation are used for water tank level behavior in the environment. The home/not home array from the CBA is used for modeling the coming and going of thecharacters during training. A simulated TOU signal is also used for an input in training. The HWH RL agent is trained over 1,573,000 timesteps or approximately 45 simulation years.

The HVAC RL Agent is designed to ensure that the temperature in the home does not exceed +/- 1° F outside of the character's preference temperature while they are home. Also, a target limitin uses per day is set that is based on the HVAC usage in the non-AI simulation in order to ensurethe RL agent operates as efficiently as possible. The agent is trained to use energy as much as possible in low cost TOU signal timesteps, however, the major priority for this agent is comfortin the home. For training, the temperature preference and home/not home arrays are importedfrom the CBA. Also, outdoor temperature and simulated TOU signal are imported. The HVAC RL Agent is trained over 2,620,800 timesteps or approximately 75 simulation years.

After the RL agents are trained and need to be used together in the real-time simulations, the RL agent environments for the HWH and HVAC RL Agents need to be modified slightly. For this

reason, completely new environments were made for these two agents that can be used specifically for real-time simulation. For HWH, the methods for determining water tank level areremoved. The Agent will receive its tank level information from the Residence Main Environment Simulator. For HVAC, the methods that calculate indoor temperature after a selected action and calculate all next temperatures for all potential actions are removed. In real-time simulation, theindoor temperature calculation based on HVAC action taken will be carried out by the Residence Main Environment Simulator, while predictions of next temperature for all possible actions is provided each timestep by the Temperature Control FC-NN. The Temperature Preference FC-NN provides the HVAC RL Agent with the predicted temperature preference of the characters throughout the current episode. The HWH and HVAC RL Agents are updated each timestep in real-time simulation by the Home or Away RNN, which provides the predicted schedule of home/not home for the remainder of the episode. All RL agents are updated each timestep inreal-time simulation by the Grid AI EMS TOU RNN, which provides an updated TOU signal schedule for the remainder of the episode. All of this information is used to update the state space of each RL agent for each timestep in a real-time simulation.

4.3 Residence Main Environment Simulator

The Residence Main Environment Simulator (RMES) (ResidenceAI_RTAgentEnv.py) is responsible for maintaining the status of the residence during real time simulation in regards to eachappliance state, character home or away, actual character temperature preference (not from Temperature Preference FC-NN which is a prediction), enable state for Dish and Laundry, showerstate, each appliance state, indoor temperature, outdoor temperature, TOU signal collected form Grid AI EMS, energy simulation for each appliance, general energy usage, and pricing. The RMESserves as the central control for the residence side in a real-time simulation. It houses all of theappliance energy simulation functions that are used in the non-AI simulator. The only differenceis that the actions to turn on or off the appliances come from the RL agents and not from the CBA or, in the case of HVAC, differences in indoor temperature and character preferences.

Each timestep, the RMES takes in the state of each appliance based on the actions of the appliance RL agents. Based on those actions, energy calculations are performed as needed. Watertank level is calculated based on appliance and shower usage. General energy use calculation is made from the home/not home input array. Final energy costs calculations are made by the RMES, although the price per kWh for each timestep is received from the Grid AI EMS in real timealong with the TOU Signal Schedule. The pricing is based on the TOU signal and will be explainedin more detail in another section.

One of the key features of the RMES is the TOU low cost signal spreading method. On days that have 18+ timesteps scheduled for a low cost TOU signal, the RMES initiates a method that sendsa delayed low-cost signal to the Laundry and HWH RL Agents. The delay is enough to ensure thatone appliance has completed a cycle before the next should turn on. This is done to prevent allthree of these agents to activate simultaneously, creating a large spike in demand and likely exceeding solar generation in that given timestep. If solar generation is exceeded and the

maximum use from battery is used, the remaining energy has to come from the grid. The TOUlow cost signal spreading method was created to “spread” the demand between these appliances across the projected timesteps with a low cost signal, ultimately leading to less grid energy used. This does not occur every day, as many days do not have 18+ low cost TOU signal timestepsscheduled.

5 Grid AI EMS

The Grid AI EMS was developed to simulate a distributed resource-enabled microgrid EMS that has the capability to respond to behaviors in grid demand in real time and project grid demandfor the remainder of a given day based on past behavior. Having this projection allows the gridEMS to maximize the energy used from the distributed sources, such as solar generation andbattery storage, and minimize the energy used from the larger main/utility grid. The Grid AI EMS consists of the following key components:

- Battery Storage and Solar Generation Simulator
- Grid Load K-Means Clustering Application
- TOU Signal Generation RNN

5.1 Battery Storage and Solar Generation Simulator

The Battery Storage and Solar Generation Simulator (BSSGS) (Grid_EMS.py) models the behaviorof a solar plus storage distributed resource. Solar generation is taken as an input, which is wasobtained from actual solar production data for Hamilton County, TN at NREL.gov/grid/solar- power-data. The battery parameters such as maximum available charge and max battery use per timestep are entered to define the constrains of the battery behavior. For this project, these values were 34.1 kWh and 1.705 kWh/timestep respectively. Grid load/demand is another input into the BSSGS.

The priority for addressing grid demand for a given timestep is in the following order: power fromsolar, power from battery storage, power from main/utility grid. Using this approach, energy from the grid is only used to meet load demand if there is not enough solar generation or battercharge available. The program does not allow battery charging from the main/utility grid, whichmeans battery can only be charged from solar generation. The program has a setting that allows for charging from grid, but this was not enabled for this project. The BSSGS saves the batterystate at each timestep, and also logs the usage from the main/utility grid and the excess solar for each timestep. Excess solar occurs when the entire grid demand is met and the battery is fullycharged in a given timestep.

The BSSGS contains a method for creating the TOU signal based on the thresholds provided in the AI System Overview section. A TOU pricing array is also created that is based on the TOUsignal generated. TOU pricing uses the TOU pricing data from California as a guide, and givesprices per kWh based on the parameters listed in Table 7:

Table 7: TOU Price Generation Parameters Based on TOU Signal

TOU Signal	TOU Price Calculation
-1	$0.01/kWh
0	Base TOU Cost for that day from California TOU Pricing Array
+1	• If the timestep is between the hours of 4:00PM – 9:00PM the maximumcost for that day from the California TOU Pricing Array • If the timestep is outside of 4:00PM – 9:00PM, the price will be the mean between the base and max cost for that day in the California TOUPricing Array

It is important to note, during real-time simulation two versions of the BSSGS are created. One is the Main BSSGS which monitors that actual grid load, solar generation, battery storage, and creates actual pricing based on the real outputs of the system. Another Virtual BSSGS is created to support the TOU Signal Generation RNN. This Virtual BSSGS takes in the actual battery state and solar generation from the Main BSSGS, but then uses the clustered load from the Grid Load K-Means Clustering Application and/or the TOU Signal Generation RNN feedback loop to simulatethe battery and solar production behavior based on the predicted load behavior from the RNN.So the Virtual BSSGS us updated with real information each timestep, but then cycles in a loopwith the TOU Signal Generation RNN using the predicted clustered load to create the TOU SignalSchedule for the remainder of the episode. This TOU Signal Schedule is then delivered to theResidence AI EMS. The process repeats itself in the next timestep.

5.2 Grid Load K-Means Clustering Application

The Grid Load K-Means Clustering Application (GLKMCA) was implemented to improve the performance of the TOU Signal Generation RNN, by allowing the load prediction to fall withingroupings. Viewing the load demand curve from the residence, the load usage can be groupedinto clusters. In order to automate this process, the GLKMCA was created. The scikit-learn Pythonmachine learning library was used to perform the K-Means Clustering applications.

The first step in any clustering approach is to identify the number of clusters that exist within adataset. An elbow plot was produced to determine the practical number of clusters for residential load (Figure 6).

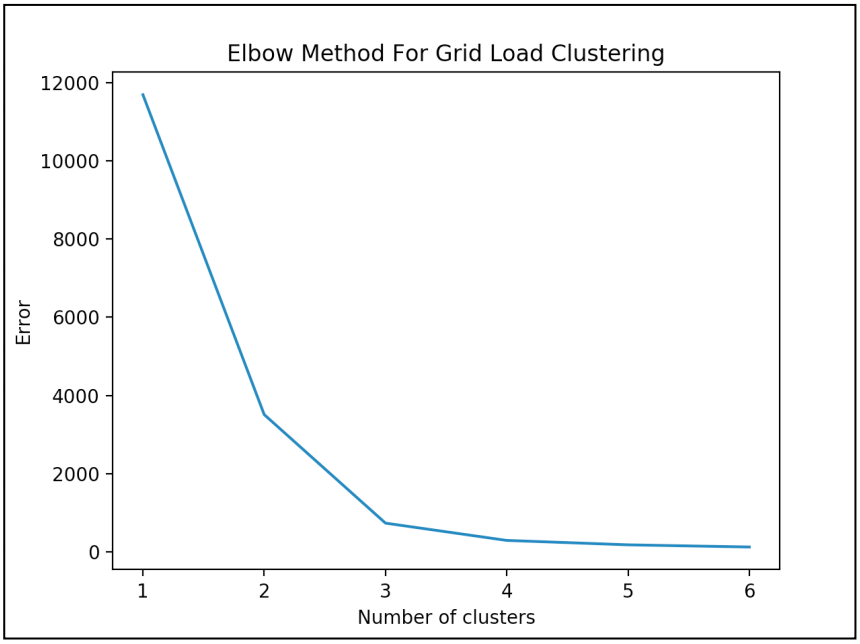

Figure 6: Elbow Plot for Grid Load Cluster Analysis

Based on Figure 6, 4 clusters should be used for grouping residential load. This is apparent because error does not decrease much beyond 4 clusters. Having four clusters creates four possible categories for load behavior to fall into, which makes for a much higher success rate for the RNN that is predicting load behavior for remaining timesteps in an episode.

The GLKMCA uses the residential grid load behavior for each scenario to create the centroids for each cluster, which are in kWh. Four clusters are used for each scenario. In real time simulation, the GLKMCA takes in the raw residential load from the Residence AI EMS (Figure 3) and sorts theload into the appropriate cluster. The clustered load from the previous 96 timesteps is loggedand is also input into Grid EMS State Space. The clustered load also fed into the Virtual BSSGS forprocessing and further input to Grid EMS State Space. When the TOU Signal Generation RNNmakes a load projection, this projection is ran in the GLKMCA to convert the predicted clusternumber into the centroid value for that cluster, which is in kWh. This value is then fed back tothe Virtual BSSGS to determine the next battery state and ultimately the projected TOU signalfor that timestep. The GLKMCA is critical for both clustering and de-clustering of residential grid load values within the Grid AI EMS workflow.

5.3 TOU Signal Generation RNN

The TOU Signal Generation RNN is really a full system that performs load behavior projection,simulates future solar generation and battery storage behavior, and generates a projected TOU signal for the remining time steps in a given episode. The load prediction RNN has the samearchitecture as the Home or Away RNN, using 96 LSTM cells with 64 neurons per cell (Goodfellowet. al, 2016). The major difference is the output function is Softmax, since the load prediction for

any one timestep is actually the cluster number (0, 1, 2, or 3). Categorical Cross-Entropy is used as the loss function with Adam optimization in back propagation during training (Goodfellow et.al, 2016).

The data used to train the load prediction RNN is taken from a simulation where the full Residence AI EMS is used with a simulated TOU signal based on the non-AI simulation energy profile (ResAI_RT_NoGridAI_SingleHome.py, ResAI_RT_NoGridAI_ThreeHome.py). This is doneto ensure that the grid load behavior of the AI-enabled residence is captured and used for training, rather than a completely different grid behavior of non-AI enabled residence. While thisis not an ideal approach it was the best course of action for this proof of concept project. An improvement to this project could be instituting daily stochastic training for all ML elements, which would allow the use of the Grid AI EMS much earlier. This was not possible given timeconstraints for this project.

The input class vector for the load prediction RNN is scalar and consists of the clustered residential load output from the simulation using the Residence AI EMS and a simulated TOUsignal. The residential load array is put into the GLKMCA, yielding an array of clustered load. Thisarray is then converted into a one-hot matrix, yielding an array of shape (34944, 4). The inputclass vector also contains the simulated TOU signal and the scalar time array. The initial inputclass vector is of shape (34944, 9). Similar to the Home or Away RNN, a timestep is paired with the clustered load in the next timestep. A loop is put in place to carry out this pairing, anotherloop is conducted to stack the training examples in groups of 96. The resulting array is of theshape (34848, 96, 13). This array is then shuffled randomly along the first axis and split into the input array (X) of shape (34848, 96, 9) and labeled output array (Y) of shape (34848, 96, 4). The training samples are then grouped into training, validation, and testing sets as consistent withother NNs in this project. The RNN is trained for 100 epochs. The accuracy results for all scenariosis 96%+.

During real time simulation the TOU Signal Generation RNN works be taking in the clusteredresidential load for the previous 96 timesteps from the GLKMCA, as well as the TOU signal from the previous 96 timesteps. These are combined with the Time Vector to create a prediction ofclustered load for the next timestep. This predicted cluster category number (0, 1, 2, 3) is thenran into the GLKMCA for de-clustering. The output is the centroid of the respective predictedload cluster, which has a value in kWh. This value is fed into the Virtual BSSGS to simulate projected battery behavior based on projected solar production. From here a projected TOU signal is created. The projected TOU signal and predicted load cluster are then cycled back into the Grid EMS State Space to be used as input for predicting load for the next timestep (Figure 3). This process is repeated until all of the remaining timesteps have a predicted load, and thereforea TOU signal generated from the Virtual BSSGS. This projected TOU Signal Schedule is then extracted from the TOU Signal Generation RNN and delivered to the Residence AI EMS. The entireprocess repeats in the next timestep of the real-time simulation (Figure 3).

6 Simulation Results and Discussions

The simulation results are provided and discussed on the website we have launched for this project. Please see http://ai-energymanagement.weebly.com/ for more details.

7 Future Improvements

Given the time constraints for this project, there were many features that could not be implemented and/or other scenarios that should be considered in continuations of this project. Considerations for future improvements are listed below.

- Custom made RL agents for appliances that incorporate more robust neural networks forpolicy development.
- RL agents that implement goal-oriented training and include methodologies such as Hindsight Experience Replay
- Custom Load Prediction RNNs for each household for multi-home grids, instead of oneLoad Prediction RNN for the entire grid.
- Scaled solar array and battery storage capacity along with a revised costing scheme forsingle-home simulations.
- Stochastic training of ML elements on a daily or weekly basis.

8 References:

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press, Retrieved from: http://www.deeplearningbook.org

Sutton, R., & Barto, G. (2018) *Reinforcement Learning*, 2nd ed., MIT Press.

Mnih, Volodymyr, Kavukcuoglu , Koray, Silver, David, Rusu1, Andrei, Veness, Joel, Bellemare, Marc, Graves, Alex, Riedmiller, Martin, Fidjeland1, Andreas, Ostrovski, Georg, Petersen,Stig, Beattie, Charles,

Sadik, Amir, Antonoglou, Ioannis, King, Helen, Kumaran, Dharshan, Wierstral, Daan, Legg, Shane, & Hassabis, Demis. (2015). Human level controlthrough deep reinforcement learning. *Nature: Research Letters*, *518*, 528 – 533. doi:10.1038/nature14236

Wang, Z., de Freitas, N., and Lanctot, M. (2015). Dueling Network Architectures for DeepReinforcement Learning. *ArXive-prints*.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. *ICLR*. Hasselt, H.v., Guez, A., Silver, D. (2015). Deep Reinforcement Learning with Double Q-Learing.
Association for the Advancement of Artificial Intelligence.