OpenGridGym: An Open-Source AI-Friendly Toolkit for Distribution Market Simulation

Rayan El Helou, *Student Member, IEEE*, Kiyeob Lee, *Member, IEEE*, Dongqi Wu, *Student Member, IEEE*, Le Xie, *Fellow, IEEE*, Srinivas Shakkottai, *Senior Member, IEEE*, and Vijay Subramanian, *Senior Member, IEEE*

Abstract—This paper presents OpenGridGym, an open-source Python-based package that allows for seamless integration of distribution market simulation with state-of-the-art artificial intelligence (AI) decision-making algorithms. We present the architecture and design choice for the proposed framework, elaborate on how users interact with OpenGridGym, and highlight its value by providing multiple cases to demonstrate its use. Four modules are used in any simulation: (1) the physical grid, (2) market mechanisms, (3) a set of trainable agents which interact with the former two modules, and (4) environment module that connects and coordinates the above three. We provide templates for each of those four, but they are easily interchangeable with custom alternatives. Several case studies are presented to illustrate the capability and potential of this toolkit in helping researchers address key design and operational questions in distribution electricity markets.

Index Terms—Distribution electricity market, open-source platform, artificial intelligence, demand response.

I. Introduction

MODERN electric grids are shifting from a centralized to a more distributed architecture. This brings up a new set of operational challenges due to the expected rapid growth of a diverse set of distributed energy resources (DERs), such as rooftop photovoltaics (PVs), electric vehicles (EVs), and storage systems at the grid edge. It also introduces more decision-makers (agents) into the picture who could strategically game the system under decentralized electricity market designs. Adding decision-makers such as DER owners, flexible loads and aggregators may significantly influence both physical and market operations. Thus, it is indispensable to understand in modern grids the implications of different market design and operational issues.

In particular, distribution grids are more susceptible to such evolution than transmission grids, as the latter is more capable at dampening intermittencies in load and generation than the former. It is necessary to focus more of our efforts on the design of distribution-level markets, and to explore alternatives to traditional approaches that rely on outdated assumptions of the distribution grid. There is a wide range of candidate market mechanisms for modern distribution grids, primarily due to the fact that much more decision-makers are involved, each with more constraining requirements than those of a transmission-level aggregator.

For example, conventional electric consumers at the distribution grids are fixed rate payers with the rate determined a-priori by the regulatory agencies. Local electric utilities plan the distribution grid capacities accordingly based on expected load growth. However, such a mechanism may be rendered ineffective due to the lack of clear incentives to encourage time-varying load flexibility at the operational stage. To address the need for new institutional design at the distribution grid level, alternatives to this pricing mechanism have been proposed in [1] where prices dispatched to *prosumers* (i.e., two-way power usage) are driven by their consumption patterns over time, measured using smart meter data. Nonetheless, demand in the short run is assumed to be inelastic. Similar market mechanisms for a wholesale market-like distribution locational marginal pricings (DLMPs) have also been introduced and conceptually investigated in the literature [2].

While there is a growing body of literature that advocates the use of DLMPs and theoretical properties of DLMPs have been investigated [3], [4], [5], concrete comparisons with alternatives to DLMPs are missing and are yet to be analyzed both theoretically and empirically. This is an important gap between the conceptualization of how market clearing and pricing should be done via DLMPs and its implementation in practice. Moreover, while game theory offers analytical tools to investigate strategic interactions on how DER, DER aggregators and flexible demands may participate in future electricity markets, without making substantial assumptions on how decision-makers strategically interact, analysis of game theoretic models is intractable in many settings [6].

Similar to the LMP calculation in bulk transmission system market operation using Optimal Power Flow (OPF), the approach and formulation have been proposed for the computation of DLMP in [7], [8] for distribution networks, using
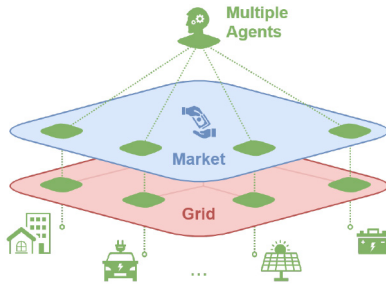
Fig. 1. Envisioned distribution grids with DERs participating in local markets.

the DistFlow power flow model for radial networks, with the objectives to minimize either power loss or generation cost. In [5], the properties of convex relaxations using the DistFlow model are investigated, and Distribution Locational Marginal Pricing (DLMP) is formulated, and results reveal the nature of DLMP distribution relative to physical constraints. In [2] the degradation cost of transformers is explicitly factored in the LMP calculation for distribution networks.

The inelastic nature of electricity demand is frequently the binding constraint of LMP-based mechanisms in OPF-based power system planning, where many ultra-high marginal prices result from the necessity to meet invariant local demand either by using expensive reserve or sub-optimal dispatch. Demand response (DR) programs can incentivize end-users in real-time to adapt their power consumption to the availability of electric power generation and delivery and introduce elasticity. References [9], [10] and [11] propose a novel DR program where end-users receive coupons as incentives to shift power consumption from peak hours to off-peak hours. This approach has been tested by conducting a case study on real users in Texas, with results that suggest that people respond positively to such incentives. Such programs can be easily implemented using our proposed simulation framework.

A. Existing Approaches to Electricity Market Simulation

There already exists a set of widely-used open-source tools for simulating physical operations of both transmission grids (e.g., MATPOWER [12] and pandapower [13]) and distribution grids (e.g., OpenDSS [14] and GridLAB-D [15]), and a proposed set of tools for electricity market simulation (e.g., AMES test bed [16]). All these tools work well under a traditional assumption of *weak coupling* between physical and market models of electric power grids. This assumption needs to be re-visited in modern grids due to the intermittence of electricity generation and consumption, as well as the likely participation of a variety of large and small agents.

In this paper, we provide a simulation toolkit that can help researchers simulate and compare the outcomes of various market mechanisms for realistic distribution grids. As implied in Figure 1, we rely on a framework that enables modular representation of grids, markets, and DER-controlling agents, which could potentially participate either in market or physical grid operations. In contrast, the existing set of tools is either not friendly to learning-based algorithms, or does not provide an easy-to-interchange modular structure which enables

experimenting with various models for both grid and market operations.

B. Our Contributions

To account for the multiplicity of agents in distribution electricity markets, we propose a new formulation where demand and supply entities are agents that submit bids or offers into the market, and prices are dispatched to them by a market operator. We develop an OpenAI Gym-like [17] benchmark for testing such multi-agent environments and invite machine learning (ML) and power systems experts to test the use of reinforcement learning (RL) to fulfill user-defined objectives. A similar effort has been made by RTE France with their Grid2Op platform, an environment popularized by the L2RPN competition [18], but the platform is restricted to physical operations, and does not include market operations. Reference [19] reviews the use of RL in electric power systems, and the aim in our proposed work is to provide a benchmark environment for distribution electricity market simulations.

Here are the key contributions of our work:
- We propose a framework to serve as a benchmark for joint market and distribution grid simulation in a competitive multi-agent setting.
- We provide an open-source Python-based user-friendly toolkit for performing simulations with trainable AI-driven agents, with use cases to demonstrate it.

The remainder of this paper is organized as follows. In Section II, we introduce our proposed Python-based package, OpenGridGym [20], and we provide a high-level overview of its architecture and user interface. In Section III, we dive deeper into the definitions of each simulation module and how those modules interact with one other to deliver a user-friendly experience. In Section IV, we present three different use cases of OpenGridGym that showcase its extensibility. In Section V, we provide concluding remarks and future directions.

II. OPENGRIDGYM

A. Motivation

As distribution grids continue to modernize, the need to reconsider traditional market mechanisms grows, and some questions need to be raised. For example, should local markets be based on peer-to-peer transactions, or should the distribution grid be governed by DLMP-based mechanisms, and what are the implications of such propositions? To answer these questions in a consistent manner, we should be able to easily integrate models of physical distribution grids with market models in a unified framework which is accessible to all researchers in this domain. Moreover, what role does artificial intelligence (AI) play in shaping the future of electricity markets? By using a programming language like Python, an entire ecosystem of AI-friendly tools is inherently available to a researcher who seeks to answer such questions.

With our proposed modular Python-based simulation package, OpenGridGym [20], it is easy to swap out market mechanisms while keeping the same physical grid model, and vice-versa, to answer questions such as the ones asked above.

The name "OpenGridGym" refers to the fact that we provide an open-source environment to exercise various grid and market designs, and to train agents to meet their objectives. By leveraging existing open-source simulation tools already developed by researchers (e.g., OpenDSS by EPRI [14]), OpenGridGym is able to account for new modalities such as real-time monitoring of substations, distributed storage, photovoltaics, electric vehicles, and more. Off-the-shelf AI-friendly tools for learning and optimization, such as PyTorch [21] and CVXPY [22], are readily available to the user, unlike with existing non-Python-based market simulators. We provide use cases in Section IV to illustrate the use of OpenGridGym.

B. Proposed Architecture and Design Decisions

In this section, we present the high-level architecture design of our simulation framework, and in the section that follows, we go deeper into how each of its components (*Grid*, *Market* and *Agents*) operate individually and collectively.

The goal of this tool is to introduce a framework for simulating electricity markets. We suggest the use of Python as the main programming language partly because it is open-source, but mainly because it hosts an ecosystem of AI- and ML-friendly tools which are readily available to the user. Our work is inspired by the well-known OpenAI Gym [17], which facilitates the simulation of Markov Decision Processes (MDPs) and testing reinforcement learning (RL) algorithms. OpenAI Gym provides the user with two main contributions: 1) A set of (empty) *base classes* which the user can fill in to represent their simulation environment, and 2) a set of use cases to demonstrate multiple examples of how such *base classes* can be filled in by the user to provide meaningful outcomes. We aim to contribute similarly in our work.

In the section which follows, we introduce the Python-based *base classes* for simulating electricity grids, market mechanisms, and various agents' behaviors which influence the two. Our design philosophy is the following. Each user of OpenGridGym should be given as much flexibility as possible over what they'd like to simulate, while simultaneously maintaining a level of consistency in the framework under which all researchers would operate. The implications are twofold. First, there needs to be minimal constraints on users' choice of grid format, market mechanism, and agents' behaviors. Second, we provide entirely Python-based easy-to-edit *base classes* which act as building blocks for the user to implement their own version of a market simulation. Additionally, we do provide a set of templates, showcased in Section IV, where we demonstrate how OpenGridGym could be used.

C. User Interface

As labelled on the left of Figure 2, there are three steps the user needs to follow to begin the simulation process (to create a grid, a market, and agents). Each of these steps involve creating their own or selecting from existing modules. The order specified in the figure is based on the principle that the physical grid is lower-level than the market which acts on top of it. Here is the order:
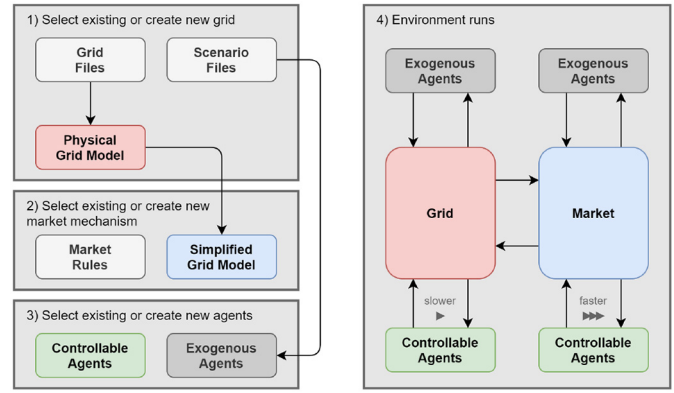

Fig. 2. User interface and simulation flowchart. The terms *slower* and *faster* indicate that market negotiations occur much more frequently than post-market-clearing interactions with the grid.

1) The user selects a grid from some case file or folder which contains the full detailed model of the physical system (indicated by the *Physical Grid Model* block). This implicitly means that the user also selects which simulator they prefer to use (e.g., OpenDSS [14]). The *Scenario Files* block indicates any exogenous input that might affect the grid state, e.g., weather and load data. Hence, it points to the *Exogenous Agents* block.

2) The user selects or creates a module to represent the market mechanism. For each market mechanism in general, there is a set of rules which dictate the prices dispatched to all participants. More specifically for electricity markets, since distribution (or transmission) grids are physical systems with constraints that need to be obeyed, any responsible market mechanism should also somehow model the grid to take those constraints into account. Such a model of the physical grid could be simple for market clearing purposes. This is indicated by the *Simplified Grid Model* block in the figure.

3) The user initializes a list of agents in the environment. Those agents are each an object in Python and they do not need to share the same decision-making policies. For each agent listed, the user either selects or creates a module to capture how they interact with either the physical grid directly or just with the market. Here is where Python's AI-friendly ecosystem can be utilized. For example, the *Controllable Agents* block refers to the fact that agents are not parametrized by a predetermined set of files (e.g., the *Scenario Files*). Rather, a learning algorithm can be associated with them so that each agent can individually seek its personal objective. This implies that such agents are effectively players in a stochastic game provided that there's more than one of them.

Once those three steps are complete, the environment can run as shown in the fourth step of the figure. We employ a two-timescale discrete-time sequential process which simulates the interplay between the metaphysical market and the physical grid. This is illustrated in Figure 3. The exact formulation of this process is shown in Algorithm 1 in the section that follows, but simply put, negotiations in the marketplace occur much more frequently than changes to the physical grid

Algorithm 1 Environment Episode

1: Instantiate the following objects in Python:
- `grid` ▷ e.g. with connection to OpenDSS
- `market` ▷ constrained by simplified grid model
- `agents` ▷ with scenarios and user profiles

2: Reset `grid` state.
3: **while** `grid` episode not complete **do**
4:   Reset `market` state.
5:   **while** `market` episode not complete **do**
6:     `agents` submit bids, offers.
7:     `market` dispatches prices and quantities.
8:   **end while** (`market` clears)
9:   `agents` submit `grid` actions.
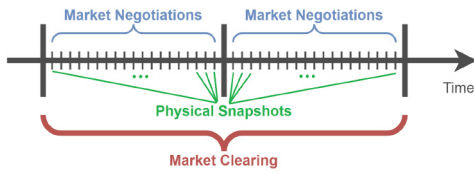10:   `grid` updates state.
11: **end while**

Fig. 3. Two-timescales for agent participation, one for negotiating in the market, and one for acting on the physical grid after the market clears.

which are governed by market clearing. Our philosophy behind this is the following. The physical grid is governed at every time step by choices made by individuals who are faced with various opportunities, namely to increase or decrease electricity consumption (or production). We assume that those choices are determined as a result of negotiations made in a marketplace. For this simple reason, we model the market with a faster time-scale. To be clear, when we say one step is taken in the physical grid, we refer to time scales closer to those of tertiary control in power grids, not to those of primary and secondary control. Transient control can still occur in between time steps marked by the term *Market Clearing* in Figure 3, but we do not impose any restrictions on such control in our proposed framework.

III. PYTHON-BASED MODULES

In this section, we define all the modules involved in the simulation process and explain how they tie together to provide a reasonable user experience. There are four modules involved in any instantiation of our proposed framework, and for each of those, we will describe the concept and show some minimal snippets of Python code to illustrate the ease of their implementation. The four modules are *Grid*, *Market*, *Agent* and *Environment*, as discussed earlier at a higher level in the previous section.

In contrast with OpenAI Gym's framework [17] for modelling discrete-time sequential processes, we do not propose that agents submit actions directly to the *Grid* object and receive observations or rewards in return. Rather, each agent sets their actions in their local memory, and the environment is expected to pull them as needed. OpenAI Gym's approach works well in single-agent settings with one environment module involved. However, when there are several players (*Grid*,
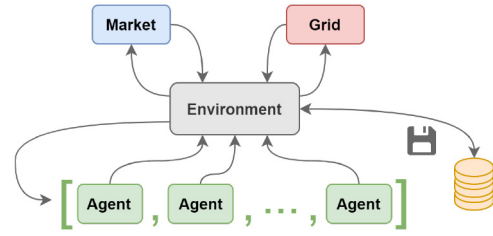

Fig. 4. Implicit access of Python objects to one another via the *Environment* object. The head of an arrow is accessible as an attribute by the object on the tail of the arrow. Note: agent-to-agent communication is inherently possible.

Market and multiple *Agents*), building an explicit communication layer is cumbersome. Our simple solution is that each of those players or objects has direct access to an object we call the *Environment*, which in turn can access all other objects, as shown in Figure 4. One of the implications of this is that all objects can implicitly talk to one another, thanks to Python's built-in pointer system.

Off-the-shelf reinforcement-learning (RL) packages can still be used to train agents under this new framework. For each agent, the user would just need to ensure that they can extract actions, observations and rewards at any time step. For example, for an agent to access observations, say voltages, from the grid, we can simply write `agent.env.grid.get_voltages()` in Python. Similarly, for the *Market* object to receive actions from some agent, the user simply calls `market.env.agent.get_market_actions()`.

There is no strictly correct way to model any of the four objects introduced in the following sections, as argued in the previous section. However, we propose that if the user follows the template we provide, then they can more consistently control experiments to compare market mechanisms with one another, agent behaviors with one another, and grid models with one another.

A. Grid

We propose that each grid object the user instantiates must have at least two functions implemented: `reset` and `step`. This is shown in the snippet code below.

```python
class CustomGrid(BaseGrid):
    def __init__(self, dss_case='',
                       scenarios=''):
        self.dss = DSS(case=dss_case)
        self.scenarios = scenarios
        ...
    def reset(self):
        self.t = -1
        ...
    def step(self):
        ...
        self.dss.solve()
        ...
```

At the beginning of each episode, the environment automatically calls `env.grid.reset()` to reset the state of the grid. This includes resetting timing, weather information, any control equipment's states, scenarios, etc. At every iteration, the environment automatically calls `env.grid.step()` to

execute all agents' actions, to solve power flow, to update its states and to store any relevant information as a result of agent's actions.

The actual model of the grid, labelled as `CustomGrid` in the snippet of code above, is either created (and filled in) by the user or selected from existing templates which we provide. For example, for a use case we show later in this paper, we rely on an IEEE 34-bus distribution grid in which the `step` function collects agents' actions and uses those to update the OpenDSS-based model accordingly.

Under this framework, leaving the `step` function empty is equivalent to simulating scenarios where agents do not influence the physical grid at all. This could be useful when experimenting with market-only interactions.

B. Market

Similar to the grid object, the market object also expects two functions to be implemented by the user: `reset` and `step`.

```python
class CustomMarket(BaseMarket):
    def __init__(self):
        ...
    def reset(self):
        self.t = -1
        ...
    def step(self):
        ...
```

That is, there is no restriction on how the market is modelled, provided that the user specifies in the `reset` function how the market initializes any states or information it may derive based on the grid, and that the user specifies in the `step` function how the market uses agents' actions to dispatch electricity prices and quantities to be consumed or produced by all participants.

As shown in Algorithm 1, this market object iterates through a sequential negotiations process, but at no point in the process is the physical grid affected. Once the market negotiations terminate, we declare that the market has cleared. Based on this, agents can then determine the actual amount to consume or produce as described in the following section.

C. Agent

The minimal requirements on agents is that they are able to state what actions they'd like to apply on the market and on the grid. During the environment's market updates, agents' actions are expected to be declared in the `set_market_actions` function. Similarly, during grid updates, agents' actions are expected to be declared in `set_grid_actions`.

```python
class CustomAgent(BaseAgent):
    def __init__(self):
        ...
    def set_market_actions(self):
        ...
    def set_grid_actions(self):
        ...
```

That is, by implementing those two functions for some agent, users would have completely modelled the behavior of said agent. Of course, those functions could internally rely on other local functions, which the user is free to create in Python. We provide examples of agents in the use cases. One example is a producer agent which submits actions in the form of supply curves that the market uses to determine optimal price and quantity dispatch.

D. Environment

Finally, the environment object, which ties the *Grid*, *Market* and *Agent* objects together is presented here. We illustrate this in Algorithm 1.

From an object-oriented programming perspective, it is redundant to explicitly create an object to represent this environment if it simply iterates over all other objects. However, the environment object can provide the user with a much cleaner and easier-to-use interface. To highlight this, we show in the snippet of code below how to set up and simulate an environment having previously defined `grid`, `market` and [list of] `agents` according to the previous sections.

```python
env = Environment(grid, market, agents)
env.reset()
for t in env.iterate():
    pass
```

We go into further details in the code's documentation about the different capabilities afforded by this style of interaction with the environment, such as the use of callbacks for example to easily save or extract data mid-simulation.

As shown in Figure 4, the environment can be accessed by all other objects and can access each of them. The figure appears to suggest that observation and control are proposed to be centralized. Indeed, we propose that in simulation, all objects should be able to access one another via a central *Environment*, whereas in practical implementation there should be explicit communication networks that restrict this.

Our justification for this is that during simulation, ultimately there is only one user, which is the person using this platform. The best experience for them involves being able to easily access everything, as enabled by this 'centralized' environment object. However, with that we must emphasize that we do not enforce that the user's implementation of the *Grid*, *Market* and *Agent* objects satisfy the user's desired outcomes. Simply put, by providing flexibility under the framework, we hand over the responsibility to the user to ensure they are implementing the simulation the way they want to. Nonetheless, as shown in the section which follows, we provide templates that can aid users in designing their own use cases.

IV. USE CASES

In the previous section, we provided a blueprint for setting up and executing distribution electricity market simulations. In this section, we provide a few use cases of OpenGridGym for two main purposes: 1) to express the variety of simulations that could be executed, both in complexity and in relation

to real-world problems, and 2) to show how OpenGridGym can be used to test and train AI-based approaches to solving problems in this domain.

Furthermore, we wish to convey to the reader that it is the simulation platform and its capabilities which we seek to highlight, rather than the use cases themselves. Neither the market mechanisms used nor the agent behaviors assumed in the use cases are suggested to be ideal. We provide those simplified use cases to reflect how users might interact with OpenGridGym to investigate the impact of possible designs and mechanisms that they may desire to experiment with.

A. Use Case 1: Topology-Induced Market Power

Here are the three goals of this use case:
- Investigate the impact of network constraints in distribution grids on electricity pricing in a competitive market.
- Explore price-responsive elastic demand as an antidote to network congestion.
- Showcase the use of OpenGridGym's framework in setting up and executing a simulation environment to enable such inquiries.

To set up this use case, we follow the blueprint described in the previous section. That is, we first define the underlying model of the physical grid. Second, we suggest a market mechanism for dispatching prices and quantities based on consumer/producer input. Third, we introduce different kinds of agents to participate in this market, and describe their behavior. Finally, we show the results of the simulation extracted from the environment object.

1) Grid Model: We rely on the standard IEEE 34 node feeder as one specific model of a physical grid, and use OpenDSS to simulate power flow corresponding to different load and generation profiles. This distribution grid model is represented by a 69 kV feeder, a substation transformer which step-down to 24.9 kV, a mixture of single- and multiphase unbalanced loads (28 in total), single- and three-phase distribution lines, capacitors and regulators.

OpenDSS offers the capability to connect directly to it using Python, and we supplement that with a user-friendly interface for the user to more seamlessly interact with it. That is, when agents submit what we referred to as 'grid actions' in this paper, OpenDSS is called to solve power flow and return the results corresponding to those actions.

For the purposes of this specific use case, we are more interested in how agents interact in the market, rather than with the physical grid. Therefore, we reduce the grid model here to single-snapshot power flow, as opposed to a time-series.

2) Market Mechanism: In today's distribution grids, most consumers of electricity participate in local retail markets where a regional load serving entity offers them prices for consumption that could remain fixed for months or even years. With the increase of power demand on distribution grids, especially now with the introduction of fast-charging electric vehicles, there needs to be either an increase in the grid's capacity or some form of demand response to counteract that.

We explore a market mechanism which models consumers as price-responsive. In this model, consumer agents submit elastic demand curves as bids to the market, and the market is expected to dispatch demand corresponding to the submitted bids to ensure, using a simplified DC power flow model, that none of the distribution lines overflow. Let q_{bus_i} represent net quantity consumed at bus i, and let $q_{\mathrm{line}_{i,j}}$ represent the quantity of power flowing from bus i to bus j. A consequence of radial topology is the following equality.

$$q_{\mathrm{line}_{i,j}} = q_{\mathrm{bus}_j} + \sum_{k \in \mathcal{C}(j)} q_{\mathrm{line}_{j,k}} \tag{1}$$

where $\mathcal{C}(j)$ is the set of buses 'children' to j (i.e., downstream from the feeder). Knowing the topology, the market optimizes over all quantities consumed/produced in attempt to enforce thermal line limit constraints.

The market object is expected not only to dispatch the amounts of power consumed or produced by each participant, but to also dispatch electricity prices. Due to page limits for this paper, we defer the detailed formulation of the price-quantity optimization problem to a separate document, but here is the short summary. Consumers submit bids as demand curves, producers submit offers as supply curves, and for each of those, we rely on a simplified parametric model to define the curves. Supply and demand curves are chosen as affine relationships between price and quantity as follows (s for supply, d for demand):

$$p_s = \frac{p_{\max} - p_{\min}}{q_{\max} - q_{\min}}(q_s - q_{\min}) + p_{\min} \tag{2}$$

$$p_d = \frac{p_{\min} - p_{\max}}{q_{\max} - q_{\min}}(q_d - q_{\min}) + p_{\max} \tag{3}$$

$$p_{\max} \geq p_{\min} \tag{4}$$

$$q_{\max} > q_{\min} \tag{5}$$

where $(p_{\max}, p_{\min}, q_{\max}, q_{\min})$ sufficiently parametrize either supply or demand curves. That is, for each bid or offer, an agent submits a quadruple of scalars to represent their participation in the market. Note that supply curves are strictly non-decreasing and demand curves are strictly non-increasing. Finally, it is the market's role to ensure that all participants are minimally satisfied while leaving no money on the table, expressed as follows:

$$(p_s, q_s) \text{ on or above supply curve} \tag{6}$$

$$(p_d, q_d) \text{ on or below demand curve} \tag{7}$$

$$\sum_d p_d q_d = \sum_s p_s q_s \tag{8}$$

The objective used in the market's optimization problem is to maximize the sum of net consumer and net producer surpluses. We know from classical economics that in cases without any network constraint, i.e., no limit on exchange of quantity between supply and demand, the optimal price-quantity dispatch is the intersection of aggregated supply and demand curves. However, in case of network constraints, the optimal solution becomes less trivial. The purpose of this use case is not to advocate for this market mechanism, but rather to test the impact of demand and supply flexibility (or

elasticity) on consumer prices. We say a consumer is less *price-responsive* the steeper their demand curve is.

3) Agent Behavior: To compare price-responsive to price-unresponsive consumers, we use the parametrization of demand curves described in the market formulation. Each consumer agent submits a quadruple of scalars $(p_{max}, p_{min}, q_{max}, q_{min})$ provided that $p_{max} \geq p_{min}$ and $q_{max} > q_{min}$, and in doing so, they declare their bid to the market. Producer agents do the same with parametrized supply curves to declare their offers to the market.

Each agent seeks to maximize their own personal rewards, which are made up of two components. For each price-quantity dispatch from the market, supply agents receive a net amount of money equal to the product of price, payed for by the demand agents. Furthermore, for each supply agent, there is an opportunity cost $c_s(q_s)$ for each quantity q_s dispatched by the market. This quantity is hidden from everyone but the agent. Similarly, for each demand agent, there is a hidden utility $u_d(q_d)$ in consuming dispatched quantity q_d. From these quantities, we define the market dispatch-based reward signal r for each agent as follows:

$$r_s(p_s, q_s) = p_s q_s - c_s(q_s) \tag{9}$$
$$r_d(p_d, q_d) = u_d(q_d) - p_d q_d \tag{10}$$

Economics theory suggests that in purely competitive markets, it is in the interest of producers to submit supply curves that most closely reflect their 'true' opportunity costs, to remain competitive. When faced with network constraints however, it is possible for producers with market power to capitalize and submit much higher prices than their cost, which we explore in this use case. Nonetheless, in general, it is left to the user to utilize Python's AI-friendly ecosystem to train this agent to maximize its rewards.

4) Environment Results: For this use case, we initialize 28 demand bids (one per load) as inelastic, and we introduce a total of 5 supply units with equal supply capacity, spread uniformly across the grid. One supply unit is located at the feeder node of the distribution network with a horizontal supply curve, i.e., fixed price for all quantities, derived from transmission-level wholesale, and assumed to be exogenous to our simulation environment. The other four supply units are price-responsive, i.e., their supply curves are increasing. Under this scenario, we observe that all consumers are dispatched a price of 4.3 ¢/kWh with no congestion in the network.

Next, we iterate through the market, where agents update their bids and offers, and we purposefully instruct consumer agents to keep their demand inelastic. The results are shown in Figure 5. The natural consequence is that prices must go up since generators increased their price offers with no objections. The figure shows us two things. First, we see that not all consumers now pay the same price ({4.6, 4.8 and around 6.9} ¢/kWh). Second, even though the feeder node remained fixed price (and cheaper than other supply), consumers still buy at more expensive prices, simply due to network congestion.

Since the network is congested, it limits the amount of cheap power deliverable to certain pockets of load in the network,
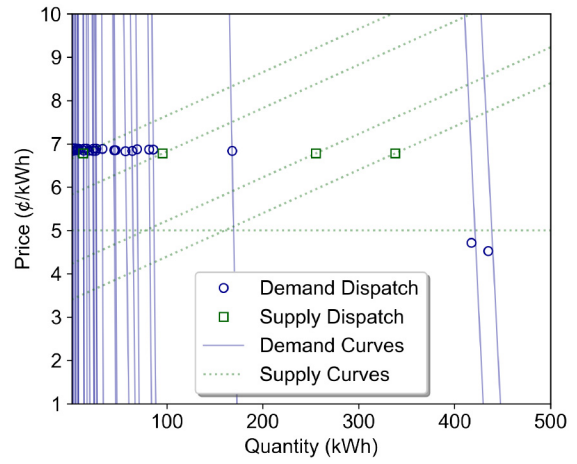

Fig. 5. Market dispatch for inelastic demand and elastic supply. Prices increase for most consumers due to line congestion.
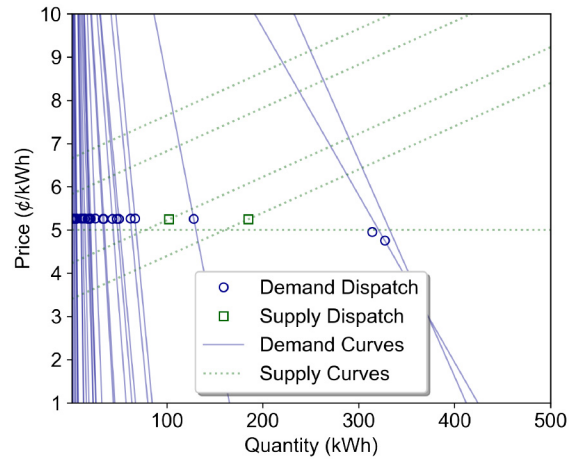

Fig. 6. More elastic demand curves lead to lower prices.

thus creating an advantage to local pricey generators. This is a fundamental property of congested radial networks. We do not wish to go deeper in this use case to analyze this market mechanism. However, we'd like to continue this example to show how consumer price-responsiveness can serve as an antidote to this topology-induced price increase.

Proceeding in the simulation, supply agents still submit relatively high prices, but the demand agents now submit more elastic demand curves. As a result, the market dispatches lower consumption quantities, even though the network remains congested. This is shown in the updated supply/demand curves of Figure 6.

A natural question that may arise is, how much demand flexibility is needed to lower the prices? To answer this, we do not change supply agents' offer for now. We just randomly select bid offers by demand agents such that some of them are flexible, and some not. We consider two scenarios, one with 8% flexible, and one with 75%. As shown in Figure 7, for the former case, the average consumer price is 6.64 ¢/kWh, whereas for the latter, the average price is 5.28 ¢/kWh.

To conclude this use case, here are three takeaway messages we wish to deliver. First, by following OpenGridGym's framework to set up this use case, we can very easily separate and

8% Demand Flexibility. Average Price: 6.64

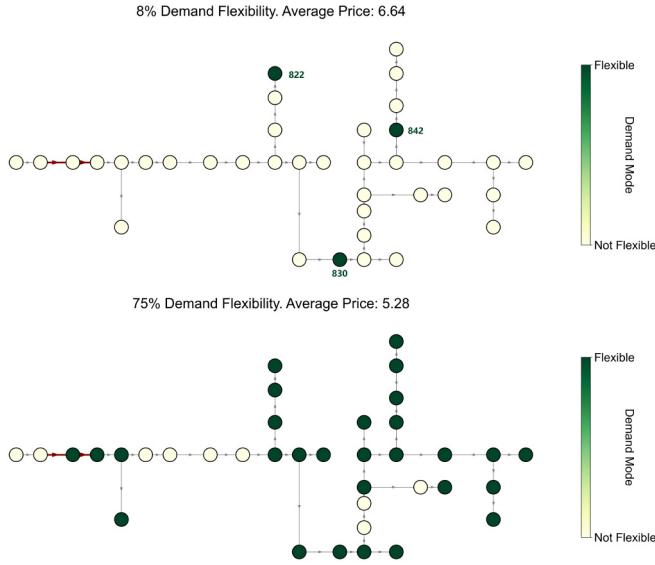75% Demand Flexibility. Average Price: 5.28

Fig. 7. Average consumer price for a congested network with different levels of demand flexibility.

debug each of the grid, market and agent simulation objects. Second, by loading a pre-defined OpenDSS-based IEEE test case, we didn't need to create the actual network, rather just specify it. This means the user can repeat this experiment on different standard grids easily, or of course create their own. Third, we were able to simulate a meaningful experiment using this framework. Namely, in this case, we were able to show that even in congested radial networks, via demand flexibility you can reduce the prices by some margin. Granted, this specific market mechanism calls for much more in-depth exploration, but here, the purpose is to provide a template for users to get an idea of how OpenGridGym works.

B. Use Case 2: Learning in Peer-to-Peer Markets

Consider a market of the sharing economy [23], [24] where market participants can switch roles between producers and consumers depending on availability of their resources. For example, PV home owners generating extra energy could shortly serve as producers by providing additional resources to the grid and they could also serve as consumers in this market when they do not produce energy from PVs. Thus, market participants are turned into prosumers who can both produce and consume resources. In this use case, we model such PV home owners as agents and consider the Peer-to-Peer (P2P) market. Here, geographic locations and weather determine which agents shift from being producers to consumers; If there are no available PV home producers, consumers can also buy the electricity from a utility provider. In this market, the role of the market maker is to match the producers and consumers so that a producer-consumer pair can negotiate to make a transaction if agreed. The objective of the market maker is to serve as a middleman between producers and consumers while respecting physical grid constraints. In this market, agents are concerned with a simple collection of tasks that they negotiate (bargain) the *agreed* price with other agents.

1) Grid Model: We use the same grid model used in Use Case 1.

2) Market Mechanism: The role of this P2P market is to match producers and consumers so that a producer-consumer pair goes through a number of negotiation steps to find an agreement on the price. If they both *agree* on the price, trade happens; otherwise they fail to make a trade. To be specific, the market maker allows a matched pair of producer and consumer to bid a number of negotiation steps, denoted by $T \in \mathbb{N}$. For each market negotiation step $t < T$, an agent (either producer or consumer) can explore to learn an optimal action, which depends on the opponent's strategies and vice versa. Given a matched pair, denote bid offers from producer and consumer by $b_{p,t}$ and $b_{c,t}$ respectively at time t. If $b_{p,t} \le b_{c,t}$, the market maker says a trade is successful. Producer p receives a reward $r_{p,t} = b_{p,t} - c_{service}$ and consumer c receives a reward $r_{c,t} = ub - b_{p,t} - c_{service}$ where $c_{service}$ is a fixed service fee in the market and ub is an utility price that consumer can alternatively purchase from. If $b_{p,t} > b_{c,t}$, then both producer and consumer receive a reward $r_{p,t} = r_{c,t} = -c_{lose}$ as a penalty when the transaction is not successful.

Lastly, we note that the market performs random matching between producers and consumers without considering agents' locations, generation amount or consumption demand.

3) Agent Behavior: We make a number of assumptions about agent behavior for simplicity of demonstration. First, we assume that all producers are homogeneous and generate the same fixed amount of electricity, say 3 kWh, and consumers purchase the same quantity. With this assumption, the market maker chooses a random matching between producers and consumers. In the absence of uniformity, the market maker can use a maximum weighted matching or other alternative matching algorithms with respect to generation amount and demand. When agents are matched by the market maker with an opponent and they negotiate. We assume that they have finitely many actions A_i to choose from in their bid offers, where $A_i = \{0, 0.5, 1, 1.5, \ldots, 10\}$. Finally, we assume that all agents implement quantal response dynamics defined as follows. When a pair of producer and consumer is matched, there are $T = 10$ episodes where each agent can interact with the other agent such that their policies are fixed within each episode and updated between episodes. The policy update is as follows:

For a fixed constant $\lambda \in (0, \infty)$ and in each episode $t \in T$, the probability of choosing action $a_i \in A_i$ given $\pi_{-i,t}$ is

$$\mathbb{P}_i(a_i, \pi_{-i,t}) = \mathbb{E}_{\pi_{-i,t}} \left[\frac{\exp(\lambda u_i(a_i, a_{-i}))}{\sum_{a_i' \in A_i} \exp(\lambda u_i(a_i', a_{-i}))} \right] \quad (11)$$

where $u_i(\cdot)$ is a reward for agent i defined in the market mechanism. Agent i does not know $\pi_{-i,t}$ because it is private information of agent $-i$, yet can approximate $\hat{\pi}_{-i,t}$ from the samples in episode $t \in [T]$ while interacting in the episode t. When π_{-i} is approximated by $\hat{\pi}_{-i}$, we can compute $\mathbb{P}_i(a_i, \hat{\pi}_{-i,t})$ for each $a_i \in A_i$ and let $\pi_{i,t+1}(\cdot) = \mathbb{P}_i(\cdot, \pi_{-i,t})$. When $\pi^* = (\pi_i^*, \pi_{-i}^*)$ is a fixed point of Eq. (11), it is called a quantal response equilibrium (QRE) [25].
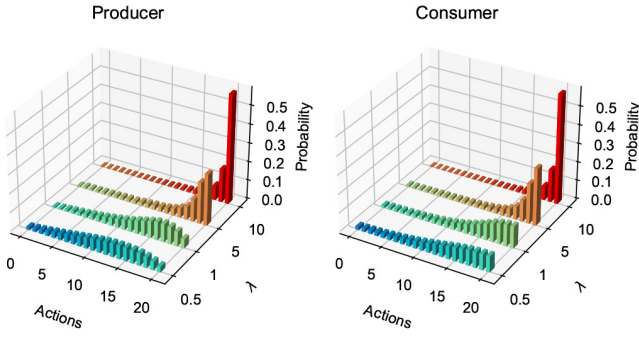
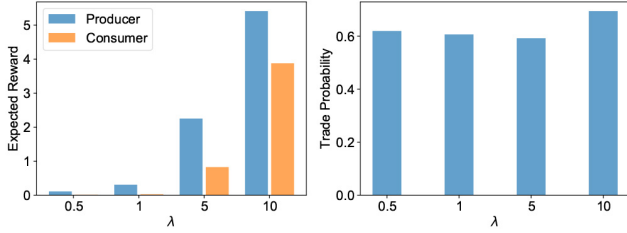Fig. 8. Action Distribution of producer and consumer.

Fig. 9. Expected Rewards and Trade Probabilities.

4) Environment Results: Using the *Grid*, *Market* and *Agent* formulation above, we implement the environment and demonstrate negotiations between a matched pair of producer and consumer. We show a multi-period negotiation between matched pairs. For each market step, the producer and consumer agents negotiate for a trade over the next several grid steps. At the time of the negotiation, generation and load forecast time series are used for the producers and consumers respectively. If a consumer cannot get enough energy from its paired producer for either supply deficiency or unsuccessful negotiation, the energy deficiency is cleared using grid power drawing from the substation at the retail price from the utility company. This formulation is able to account for the thermal inertia of loads across different market intervals and uncertainty of future renewable generation after the power transaction settlement.

The action distributions of a matched pair of producer and consumer are shown in Fig. 8 for each $\lambda \in \{0.5, 1, 5, 10\}$ where each distribution is averaged over 5 producers and 5 consumers respectively. It is observed that distribution skews towards higher actions for higher λ while distribution shifts towards a uniform distribution for lower λ. Similarly, it is also observed the expected rewards are higher for higher λ whereas the trade probability is mostly constant for different λ. Note that without grid constraints binding, there are many pure Nash equilibria in our bargaining model where trade happens. The QRE for large λ values approximates a Nash equilibrium.

It is also noticed in numerical experiments that under certain operating conditions, the settled bilateral transactions cannot be realized in the grid step. In radial distribution networks there is usually one electric path between producers and consumers at different buses, and when the power being traded exceeds rating of any line in the grid, the power flow results become infeasible. Under such conditions, market transactions can potentially impact the operation of physical distribution grid. Our platform can be used to study this impact.

C. Use Case 3

The third use case attempts to establish a general comprehensive market structure for future distribution systems where prosumers in the network trade their power generation and consumption with fellow prosumers or with the transmission grid. There are three roles for any prosumer to participate in the market: as a power producer, power consumer or demand response provider. Power producers are able to inject net positive real power into the network with distributed energy sources such as PVs, diesel engines or charged batteries. Power consumers have net negative real power capacity that need to be supplied by the transmission grid or other producers. In the case of network congestion which results in high price or insufficient supply, demand response providers at certain parts of the network can voluntarily reduce their power consumption in exchange for a profit, which is paid for by other consumers that benefit from such reduction.

1) Grid Model: The IEEE 37-node feeder benchmark system is used in this case study. This system is a 4.8kV mid voltage distribution with around 2200kW total load. This network is characterized by a all-delta configuration with one large three-phase load and many spot single-phase loads. The system information is automatically extracted and processed using OpenGridGym.

2) Market Mechanism: Using our proposed framework, we demonstrate a possible market mechanism for distribution system operators (DSOs) that utilizes the concept of Distribution Locational Marginal Prices (DLMPs). As with LMP's in current transmission systems, power producers submit a convex cost function that maps their net power production to desired revenue. The DSO then runs security-constrained optimal power flow (SCOPF) to determine the exact dispatch for every producer in the network that minimizes the total energy cost. A distinctive feature in distribution systems compared to transmission systems is the existence of an infinite source. Most if not all distribution networks are connected to the transmission system through a transformer in a substation whose capacity rating is large enough to supply the entire distribution network. However, the price to use power from the transmission system can also change with time, as the price may be affected by the transmission system LMP. The detailed problem formulation of SCOPF is as follows:

$$\min_{P^G, P^D} LMP^{source} max\left(0, P^{source}\right)$$
$$+ c_{gen}\left(P^g\right) + c_{DR}\left(\overline{P^d} - P^d\right) \quad (12)$$

subject to:

$$P^{source} = 1^T \left(P^d - P^g\right) \quad \text{(power balance)} \quad (13)$$
$$P^g_{min} \leq P^g \leq P^g_{max} \quad \text{(generator limit)} \quad (14)$$
$$0 \leq P^d \leq \overline{P^d} \quad \text{(demand response)} \quad (15)$$
$$\Delta p = \Delta P^g - \Delta P^d \quad \text{(bus injections)} \quad (16)$$
$$\Delta f^{line} = H \Delta p \quad \text{(power transfer)} \quad (17)$$
$$- f_{max} \leq f^{line} \leq f_{max} \quad \text{(line flow limits)} \quad (18)$$

where c_{gen} and c_{DR} are the cost functions of distributed generators and demand response providers; P^{source} is the total
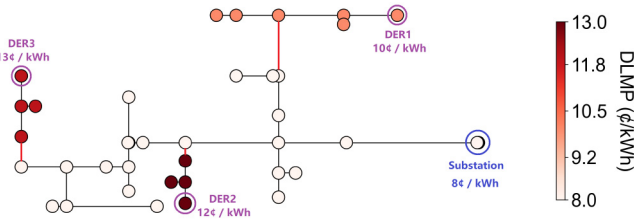
Fig. 10. Distribution of DLMPs in the IEEE 37-Node Feeder.

power drawn from the substation transformer; P^d and P^g are the power consumption and generation at each bus, while $\overline{P^d}$ is the initial load value before demand response; f^{line} is the real power flow in lines and must not exceed the ratings; H is a matrix computed from the network topology that maps net real power injections at each bus to real power flows at each line.

3) Agent Behavior: In a learning problems, the cost curves (which may be parameterized as polynomial of piecewise functions) of power producers and demand response providers are determined by agents. During every market clearance interval, each agent observes market information disclosed by the DSO and determine the parameters of its cost curve, c_{gen} or c_{DR}. After the DSO solves the SCOPF problem, each agent receives a dispatch that specifies how much power they are allowed to produce and consume. The reward of each agent is then calculated based on their net energy consumption, production or reduction and the DLMP at their bus.

4) Environment Results: OpenGridGym provides an implementation using CVXPY [22] and NetworkX [26]. At each market step, the DSO uses the agents' input to compute a DLMP for every load bus in the network, which is then used to calculate the cost and revenue of all market participants. An example of DLMP distribution in a simple radial network is shown in Fig. 10.

V. Conclusion

This paper presents a simulation platform, OpenGridGym, for scalable multi-agent market simulation for future distribution systems. This platform is open-source and based on user-friendly Python toolkit. This platform could serve as a benchmark for the research community to simulate and analyze the outcome of various market mechanisms with direct access to AI-friendly ecosystem via Python.

Building upon this open-access toolkit, many interesting research questions in distribution market design and outcome could be quantitatively analyzed. To showcase OpenGridGym, we present different use cases and demonstrate how users can easily integrate trainable AI-driven agents into their simulation. Future work on OpenGridGym includes expanding the use cases to provide users with even more templates to follow, and to assist the design of alternative market mechanisms to address challenges faced in modern distribution grids.

References

[1] A. Venkatraman, A. A. Thatte, and L. Xie, "A smart meter data-driven distribution utility rate model for networks with prosumers," *Utilities Policy*, vol. 70, Jun. 2021, Art. no. 101212.

[2] P. Andrianesis and M. Caramanis, "Distribution network marginal costs: Enhanced AC OPF including transformer degradation," *IEEE Trans. Smart Grid*, vol. 11, no. 5, pp. 3910–3920, Sep. 2020.

[3] L. Bai, J. Wang, C. Wang, C. Chen, and F. Li, "Distribution locational marginal pricing (DLMP) for congestion management and voltage support," *IEEE Trans. Power Syst.*, vol. 33, no. 4, pp. 4061–4073, Jul. 2018.

[4] A. K. Zarabie, S. Das, and M. N. Faqiry, "Fairness-regularized DLMP-based bilevel transactive energy mechanism in distribution systems," *IEEE Trans. Smart Grid*, vol. 10, no. 6, pp. 6029–6040, Nov. 2019.

[5] A. Winnicki, M. Ndrio, and S. Bose, "On convex relaxation-based distribution locational marginal prices," in *Proc. IEEE Power Energy Soc. Innov. Smart Grid Technol. Conf. (ISGT)*, 2020, pp. 1–5.

[6] S. Bose and S. H. Low, "Some emerging challenges in electricity markets," in *Smart Grid Control*. Cham, Switzerland: Springer, 2019, pp. 29–45.

[7] L. Gan, N. Li, U. Topcu, and S. H. Low, "Optimal power flow in tree networks," in *Proc. 52nd IEEE Conf. Decis. Control*, 2013, pp. 2313–2318.

[8] L. Gan, N. Li, U. Topcu, and S. H. Low, "Exact convex relaxation of optimal power flow in radial networks," *IEEE Trans. Autom. Control*, vol. 60, no. 1, pp. 72–87, Jan. 2015.

[9] H. Ming, B. Xia, K.-Y. Lee, A. Adepoju, S. Shakkottai, and L. Xie, "Prediction and assessment of demand response potential with coupon incentives in highly renewable power systems," *Prot. Control Mod. Power Syst.*, vol. 5, pp. 1–14, Apr. 2020.

[10] B. Xia et al., "EnergyCoupon: A case study on incentive-based demand response in smart grid," in *Proc. 8th Int. Conf. Future Energy Syst.*, 2017, pp. 80–90.

[11] J. Li et al., "Energy coupon: A mean field game perspective on demand response in smart grids," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2015, pp. 455–456.

[12] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[13] L. Thurner et al., "Pandapower—An open-source python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Trans. Power Syst.*, vol. 33, no. 6, pp. 6510–6521, Nov. 2018.

[14] R. C. Dugan and D. Montenegro, *The Open Distribution System Simulator^{TM}(OpenDSS), Reference Guide*, Electr. Power Res. Inst. (EPRI), Palo Alto, CA, USA, 2018.

[15] D. P. Chassin, K. Schneider, and C. Gerkensmeyer, "GridLAB-D: An open-source power systems modeling and simulation environment," in *Proc. IEEE/PES Transm. Distrib. Conf. Expo.*, 2008, pp. 1–5.

[16] L. Tesfatsion. "The AMES wholesale power market test bed." 2022. [Online]. Available: http://www2.econ.iastate.edu/tesfatsi/AMESMarketHome.htm

[17] G. Brockman et al., "OpenAI gym," 2016, *arXiv:1606.01540*.

[18] A. Marot et al. "L2RPN: Learning to run a power network in a sustainable world NeurIPS2020 challenge design." Jun. 2020. [Online]. Available: https://www.public.asu.edu/~yweng2/Tutorial5/pdf/111.pdf

[19] M. Glavic, "(Deep) reinforcement learning for electric power system control and related problems: A short review and perspectives," *Annu. Rev. Control*, vol. 48, pp. 22–35, Jan. 2019.

[20] R. E. Helou, K. Lee, D. Wu, L. Xie, S. Shakkottai, and V. Subramanian. "OpenGridGym." 2022. [Online]. Available: https://github.com/tamu-engineering-research/OpenGridGym

[21] A. Paszke et al., "Automatic differentiation in PyTorch," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, 2017, pp. 1–4.

[22] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd, "A rewriting system for convex optimization problems," *J. Control Decis.*, vol. 5, no. 1, pp. 42–60, 2018.

[23] B. Xia, S. Shakkottai, and V. Subramanian, "Small-scale markets for a bilateral energy sharing economy," *IEEE Trans. Control Netw. Syst.*, vol. 6, no. 3, pp. 1026–1037, Sep. 2019.

[24] R. Henriquez-Auba, P. Hidalgo-Gonzalez, P. Pauli, D. Kalathil, D. S. Callaway, and K. Poolla, "Sharing economy and optimal investment decisions for distributed solar generation," *Appl. Energy*, vol. 294, Jul. 2021, Art. no. 117029.

[25] R. D. McKelvey and T. R. Palfrey, "Quantal response equilibria for normal form games," *Games Econ. Behav.*, vol. 10, no. 1, pp. 6–38, 1995.

[26] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. 7th Python Sci. Conf.*, Pasadena, CA, USA, 2008, pp. 11–15.

Rayan El Helou (Student Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from Ohio State University, Columbus, OH, USA. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. His research interests include power systems dynamics, modeling and control of smart grids, and reinforcement learning.
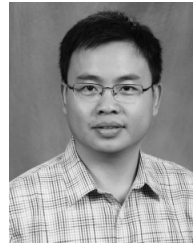
Le Xie (Fellow, IEEE) received the B.E. degree in electrical engineering from Tsinghua University, Beijing, China, in 2004, the M.S. degree in engineering sciences from Harvard University, Cambridge, MA, USA, in 2005, and the Ph.D. degree from the Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA, in 2009. He is currently a Professor with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX, USA. His research interests include modeling and control of large-scale complex systems, smart grids application with renewable energy resources, and electricity markets.
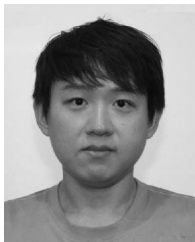
Kiyeob Lee (Member, IEEE) received the B.S. and Ph.D. degrees in electrical and computer engineering from Texas A&M University, College Station, TX, USA, where he is currently a Postdoctoral Researcher with the Department of Electrical and Computer Engineering. His research interests include modeling and control of demand response, electricity markets, and hosting capacity analysis.

Srinivas Shakkottai (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana Champaign in 2007.

He was a Postdoctoral Scholar of Management Science and Engineering with Stanford University in 2007. In 2008, he joined Texas A&M University, where he is currently a Professor of Computer Engineering with the Department of Electrical and Computer Engineering. His research interests include caching and content distribution, wireless networks, multi-agent learning and game theory, and network data collection and analytics. He was a recipient of the Defense Threat Reduction Agency Young Investigator Award in 2009, the NSF CAREER Award in 2012, and the Research Awards from Cisco in 2008 and Google in 2010. He received the Outstanding Professor Award in 2013, the Select Young Faculty Fellowship in 2014, and the Engineering Genesis Award at Texas A&M University in 2019.

Dongqi Wu (Student Member, IEEE) received B.S. degree in electrical engineering from Purdue University and the Ph.D. degree in electrical engineering from Texas A&M University. He works with Guangdong Electric Power Design Institute, China Energy Engineering Group, Guangzhou, China. His dissertation topics focused on power system hazard study and mitigation. His current interest include energy and economic development planning, low-carbon power system design, and machine learning application.

Vijay Subramanian (Senior Member, IEEE) received the Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign, Champaign, IL, USA, in 1999. He worked with Motorola Inc., The Hamilton Institute, Maynooth, Ireland, for many years, and also with the EECS Department, Northwestern University, Evanston, IL, USA. In Fall 2014, he started in his current position as an Associate Professor with the EECS Department, University of Michigan, Ann Arbor. From 2022 to 2023, he is visiting UIUC hosted graciously by CSL and also the ECE Department. His research interests are in stochastic analysis, random graphs, multiagent systems, and game theory (mechanism and information design) with applications to social, economic, and technological networks.