A CONVERSATIONAL HUMAN-COMPUTER INTERFACE FOR SMART ENERGY SYSTEM SIMULATION ENVIRONMENTS

Gabriel Dengler
Peter Bazan
Reinhard German

Pooia Lalbakhsh
Ariel Liebmann

Laboratory of Computer Networks
and Communication Systems
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstr. 3
Erlangen, BY 91058, GERMANY

Department of Data Science and AI
Faculty of Information Technology
Monash University
Clayton, VIC 3168, AUSTRALIA

ABSTRACT

This paper introduces a conversational framework that enhances the usability of smart energy system simulations. This study is centered around OpenAI's Generative Pre-trained Transformer (GPT), a fine-tuned conversational model that allows users to communicate with the system in a natural way. Therefore, users can describe their simulation scenarios in plain language and GPT seamlessly translates these descriptions into Python scripts, used as inputs to the simulation environment, in our case, AnyLogic Simulation Software. Our framework is based on the i7-AnyEnergy core framework to compute distribution flows and relevant statistics. The proposed human-machine interface facilitates and accelerates simulation modeling, as demonstrated through the two scenarios we have provided in this paper. Overall, our conversational framework has the potential to significantly improve the user experience of smart energy system simulation environments. By simplifying the interaction between users and complex simulation models, we enable users to obtain valuable insights rapidly and more easily.

1 INTRODUCTION

Simulation modeling has become a powerful tool for decision-making in the design and management of complex systems and processes. This technique enables the study, analysis, and evaluation of situations that would be otherwise difficult, costly, or in some cases impossible to explore. In today's fast-paced, data-oriented business environments, simulation modeling has become an essential planning tool for managers, designers, and engineers across a range of domains, from public health (Zhang et al. 2023) to natural disasters (Song et al. 2016), defense applications (Lalbakhsh et al. 2018), traffic management (Long et al. 2023), etc. Simulation modeling strategies and tools are now being used as core cogs of other higher-level decision support systems such as digital twins (Jones et al. 2020), and they are being used on a broad scale, from the microscale (Pan et al. 2023) to the macroscale (Wittor and Gaspari 2023), to address different challenges.

The evolution of power grid technology has been motivated not only by the goal of increasing energy security and decreasing carbon emissions but also by the need to address the urgent issue of climate change (He et al. 2023). The shift in priorities has led to a surge of interest and investment in the research and development of smart energy systems and the transition from conventional power systems to microgrids, which in turn introduces new challenges, such as the integration, management, and optimization of storage devices and renewable energy sources to the grid (Saeed et al. 2021).

Simulation tools are extensively employed across various domains of power systems, including power production, distribution, and consumption. They provide users with the ability to create and evaluate diverse scenarios, optimize performance, and predict outcomes to improve the efficiency of various aspects of the power grid. However, the accuracy and effectiveness of simulation results depend significantly on the expertise and skills of the user in developing scenarios. Additionally, each simulation environment has its own set of limitations due to its architecture, development language, interoperability, and target aspect of the power grid it is designed for. Therefore, it is usually necessary for the user to gain a thorough understanding of the simulation environment before starting to be productive with it.

While technical limitations can be a challenge to the effectiveness of simulation tools, poorly designed and complex human-computer interfaces can also hinder the full potential of simulation environments. This issue is particularly relevant when users have to rely on Command-Line (CL) interfaces or use a specific high-level or proprietary scripting language (King and ChatGPT 2023). Such limitations can slow down the development process, limit the usability of the tool, and prevent users from utilizing its capabilities to the fullest extent. This study aims to address this challenge. Initially, the only way to communicate with computers was through machine language, a string of zeros and ones. However, with the emergence of advanced Large Language Models (LLMs), computers can now understand conversations and natural prompts, turning the tables on human-computer communication (Geerling et al. 2023).

This study introduces a novel conversational simulation framework for enhancing the end-user experience considering simulation environments. This framework is specifically applied to smart energy systems, building upon the i7-AnyEnergy framework, which is available at https://github.com/cs7org/i7-AnyEnergy as open-source software (Bazan 2017). To the best of our knowledge, this is the first study introducing a conversational simulation framework for energy systems. Despite the hype surrounding LLMs and their applications, particularly in areas such as AI ethics (Liebrenz et al. 2023), social privacy (Tlili et al. 2023), and plagiarism (Anders 2023), there is still limited research on its practical implementations. In terms of real-world applications, Microsoft's robotics control framework, powered by the OpenAI's GPT model (Vemprala et al. 2023), serves as an inspiring example that has motivated this research. Furthermore, there are ongoing discussions regarding the potential of LLMs for industrial applications such as intelligent vehicles and autonomous driving (Du et al. 2023) and sustainable manufacturing (Rathore 2023).

Our framework leverages the power of Generative Pre-trained Transformer (GPT), an advanced conversational interface, to enable users to describe simulation scenarios in natural language, without requiring any technical expertise of the underlying system. At the time of writing this paper, we resorted to the ChatGPT service (https://chat.openai.com/) which was based on a GPT-4 series model. A high-level function library provides GPT with the necessary information to create a variety of scenarios according to the user inputs. Using this information, GPT can interpret the scenario described by the user and map it into an i7-AnyEnergy construct, which is then dynamically built and executed under the AnyLogic simulation environment, specifically the AnyLogic 8.7.12 University Researcher Version (https://www.anylogic.com/downloads/).

2 MODELING A CELLULAR ENERGY SYSTEM

In this study, we resort to the modeling of energy systems based on the so-called Cellular Energy System (CES) which was originally introduced by the Verband der Elektrotechnik (VDE) (German for "Association for Electrical Engineers") (VDE Verband der Elektrotechnik Elektronik Informationstechnik e.V. 2019) and aims to overcome the challenges that occur by an increasing amount of decentralized energy production and storage in carbon-neutral energy systems. The main idea of a CES is to combine coherent elements of energy systems at different levels into a single Energy Cell (EC), whereby a hierarchical structure is formed. Furthermore, ECs can also communicate on the same level, leading to neighborhood connections.

Since the proposal of the VDE gives only some rough concepts instead of precise modeling instructions for a simulation environment, we follow a concrete modeling and simulation approach as presented in (Dengler et al. 2022). According to this approach, an EC can be considered as:
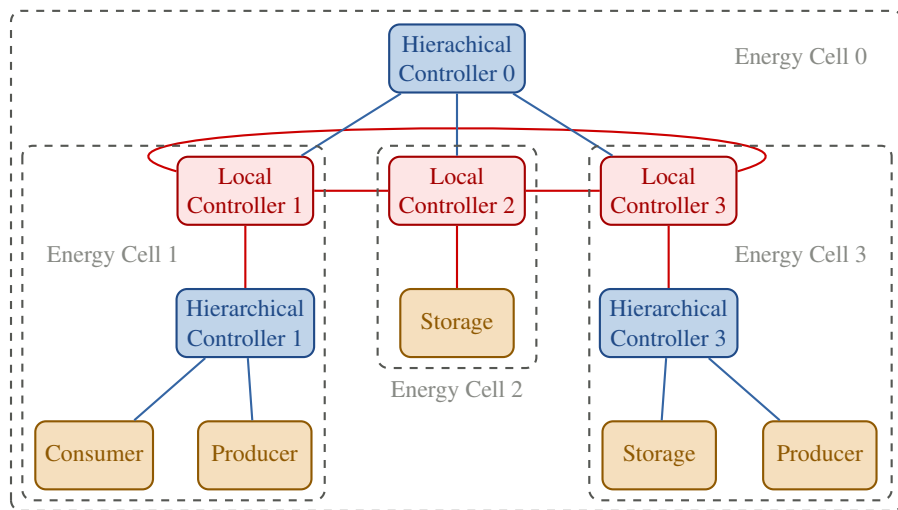
Figure 1: Small example of a CES architecture as used in this paper.

- • an atomic component, such as a consumer, producer, or storage unit,
- • a Hierarchical Controller (HC) with an arbitrary amount of aggregated subordinate ECs,
- • or a Local Controller (LC) that has exactly one underlying EC and any number of neighboring LCs.

Figure 1 presents an illustrative model that this study is based on. The figure showcases the various components and their corresponding use cases. This visualization shows how a pair of HCs and LCs, for instance, can be used to represent hierarchical and neighborhood connections. We clearly differentiate between HCs and LCs. HCs are introduced specifically to aggregate multiple subordinate ECs, whereas in cases where such aggregation is unnecessary (e.g., EC 2 in the figure), this modeling approach allows for the omission of the HC component. Additionally, the load-balancing process is semantically divided into two distinct parts: one for hierarchical relations and another for neighborhood relations. This division enables independent customization of the algorithmic procedures for each, granting the ability to fine-tune and optimize load balancing individually.

The chosen approach in this work offers two-fold benefits. Firstly, it facilitates the implementation of different load-balancing strategies such as self-sufficiency, enabling a realistic modeling of future microgrids. However, it is important to note that this work does not delve into the calculation of energy balancing based on the given model, as the focus lies solely on the system's structure. Secondly, the object-oriented representation of CES components, including atomic components and controllers, provides an easily understandable Application Programming Interface (API) definition for the LLM, simplifying the creation of scenarios.

It is worth acknowledging that there are many other approaches addressing the CES concept, for example, the definition by Björn Uhlemeyer et al. (Uhlemeyer et al. 2021) or the *C/Sells* project (Haller et al. 2020). Additionally, there are similar studies that follow the idea of recursively constructed energy systems, named *Autonomous Energy Grids* (Kroposki et al. 2020) or *Fractal Grids* (Retière et al. 2017).

3 SCENARIO GENERATION

Figure 2 presents the overall pipeline of the suggested approach and the involved components. We start with the high-level description from the user that will be given to an LLM, which was prepared with an initial preamble. The model now creates a Python script that assumes the availability of functions as defined in the preamble. With an implementation of these functions from the preamble we can execute this created code and generate the input files for the actual simulation environment. In our case we are dealing with Comma-separated values (CSV) files as input for i7-AnyEnergy (Bazan 2017), a second-layer framework
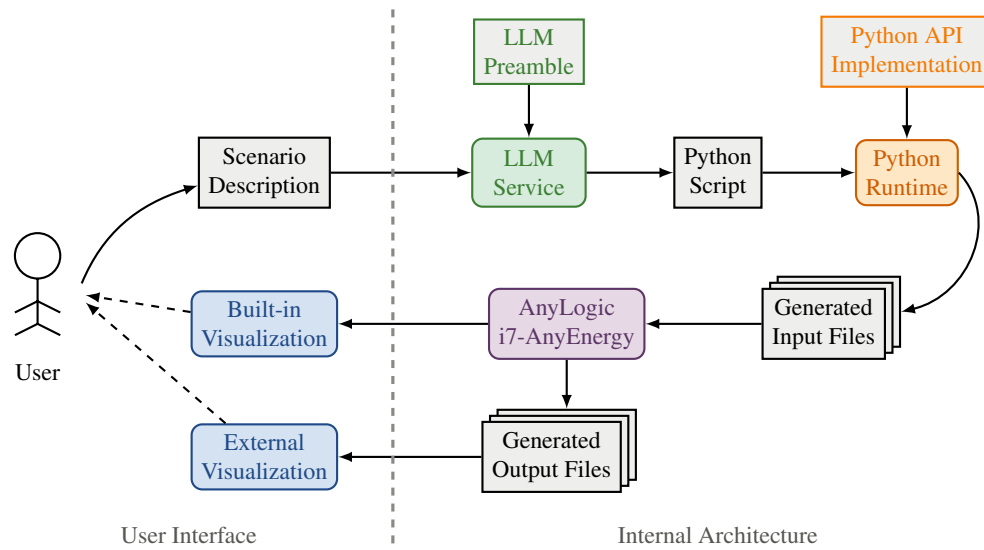
Figure 2: Proposed architecture.

to the simulation environment AnyLogic. The output can be visualized in two ways: the first is to use the built-in visualization feature from AnyLogic directly, and the other one is to make use of the generated output files (given in CSV format) from i7-AnyEnergy and present them in a human-understandable way.

With this setup, the user only needs to formulate the desired scenario in a high-level description without the necessity to understand the individual details of the used frameworks. This makes huge simulation frameworks far more accessible. The reason why to use an intermediate API instead of letting the LLM create the individual input files directly is to give the LLM a framework that is as easy to use as possible. The input files of i7-AnyEnergy are rather complex and need much more text to create than using a Python API. This Python API can give an easy interface to construct a CES from a semantic perspective and is based on a programming language that the LLM is already aware of. Furthermore, this intermediate step can decrease the run-time of the LLM input generation for large scenarios substantially. This applies especially to sections in the input data where the created code can make use of widely used programming techniques such as functions, loops, or recursion. Although we use OpenAI's GPT for the developed prototype, we can easily exchange it with another language model like Google Bard (https://bard.google.com/).

We would also be able to make use of an LLM as an intermediate layer between the generated outputs of the simulation and the presentation to the user, e.g., by letting the LLM extract the important details of the output. However, our paper does not focus on this part, as we think that classical analysis methods are better suited for this application. Compared to the high-level input description from the user, the output files are given in a structured way, so they can easily be analyzed by a program. One potential scenario where an LLM could be useful is when the user desires to extract specific information from the output. In such cases, the LLM could serve as a translation layer, facilitating the transformation of user requests into alternative representations, such as the integration of the Wolfram Language in ChatGPT (Wolfram 2023), but this is a topic out of the focus of this paper.

3.1 Communication with the Large Language Model

As suggested earlier, we use an LLM as a tool to translate the high-level description of the scenario from the user to a corresponding Python code. To achieve this, we provide the LLM with an initial message comprising three essential components:

1. The first part entails explaining the fundamental concept of a CES and introducing the terminology within this domain.

2. Following that, we make the model familiar with the goal of writing a small Python program. We introduce the relevant data types and deliver a detailed Python API description.
3. Lastly, we specify that any uncertainties regarding the user input should be ignored and we seek confirmation from the model to ensure everything has been understood correctly.

The concrete input message is given as follows:

Preamble to GPT and corresponding answer

The following task is to create input examples for a simulation environment for Cellular Energy Systems. The basis of the Cellular Energy System is the Energy Cell. An Energy Cell can be an atomic component like a consumer, producer, storage unit, or an aggregation of multiple underlying Energy Cells with the help of a controller. Several Energy Cells can be combined to form a hierarchically superordinate Energy Cell. These hierarchically superordinate Energy Cells can in turn be combined and thus form a complex hierarchy. Furthermore, there exist neighborhood connections that allow additional energy exchange between controllers. Note to add no circular dependencies for controllers, e.g., a child of a controller cannot be a neighbor of a controller at the same time.

The goal is to write a Python program for any of the following inputs that give a high-level description of the wanted scenario.

The central data type is `controller` which describes a controller instance that aggregates an underlying number of Energy Cells. Do not assume that there exist any class fields or functions for the `controller` data structure. Only assume that the following Python functions are available to define the Cellular Energy System structure:

(Python API description as explained in Section 3.2)

Note to use `generate_root()` exactly once at the beginning of the program execution and append every directly connected controller to it. For any other layer, recursively append the controllers to form a tree-like structure.

If anything is unclear regarding the user input description, try to reasonably fill the information gap by yourself.

Are you ready?

- -

Yes, I'm ready. Please provide the input scenario you want to create for the Cellular Energy System simulation environment.

It becomes apparent that we do not distinguish here between HCs and LCs and only see a pair of HC and LC as a controller. The reason for this decision is to simplify the model architecture to reduce the risk that the LLM violates some constraints of the model, e.g., allowing an LC to have more than one subordinate EC. The underlying API implementation automatically creates an HC-LC pair for every controller, regardless of the number of subordinate ECs or neighbor ECs, but it could also detect if an HC or an LC is necessary and only insert them if necessary.

It is important to note that we explicitly state at some points that the LLM should not assume the existence of functions or fields available to the central data type `controller`. It prevents the LLM from attempting to utilize non-existent fields in controllers (e.g. `controller.children`), which could lead to errors during code execution. To address any potential information gaps or uncertainties, we instruct the LLM to autonomously fill in the missing information. This ensures the generation of a reasonable output and facilitates the smooth execution of subsequent steps in the architecture's pipeline. However, the main

drawback of this design choice is that the user is not informed about the assumptions made by GPT, which can lead to undefined behavior for poorly designed inputs. To accompany these issues, we could exchange this instruction with a request to additionally state uncertainties, giving the users some hints on how to adjust their input.

3.2 Delivered Python API

The Python API makes use of the recursive structure of the considered CES modeling approach. The hierarchical connections can be defined by starting with the root controller (`generate_root()`) and then adding subordinate controllers step by step (`add_sub_controller()`). As a final step, we can add atomic components like producers, consumers, and storage units. After the creation of the controllers, the neighborhood connections can be added (`add_neighborhood_connection()`). Due to the large number of parameters for the storage units, we define default values for each of them in case not all of them are specified by the user. The depicted example values should represent an initially half-charged small battery that loses some energy during operation. Furthermore, there exists a function to define the duration of the simulated time (`set_simulation_duration()`) and one to confirm the end of the generation script (`write_output()`).

The detailed API description, which is also given to the LLM, is as follows:

Python API description in preamble given to GPT

-   `set_simulation_duration(hours)`:
    Sets the duration of the simulation run in `hours`. When this function is not called, a duration of 24 hours of simulated time is assumed.
-   `generate_root()`:
    Generates the root controller and returns it. This function has to be called exactly once in the beginning, whereas every Energy Cell is a child of that root controller.
-   `add_sub_controller(controller)`:
    Creates a subordinate controller and adds it to the current controller. This function returns the newly created controller.
-   `add_neighborhood_connection(controller1, controller2)`:
    Creates a neighborhood connection between the controllers `controller1` and `controller2`. `controller1` and `controller2` have to be created beforehand. There should be at most one connection between two controllers.
-   `add_sub_consumer(controller, values)`:
    Adds a consumer to a controller. The `values` array is a Python list that contains the generation specifying the generation in kW with an accuracy of five minutes. So, for a 24-hour day, the `values` array contains $24 \cdot 12 = 288$ entries.
-   `add_sub_producer(controller, values)`:
    Adds a producer to a controller. The `values` array is a Python list with the same behavior as in the function `add_sub_consumer`.
-   `add_sub_storage(controller, capacity=3.0, start_soc=50.0, max_charge=3.0, max_discharge=3.0, min_charge=0.99, min_discharge=0.99, leak_fraction=0.1, charge_eff=0.9, discharge_eff=0.9)`:

Adds a storage unit to a controller. The `capacity` defines the maximum state of charge in kWh. `start_soc` defines the load of the storage unit in percent at the beginning. `max_charge` and `max_discharge` define the maximum charge and discharge speed in kW, `min_charge` and `min_discharge` define the minimum charge and discharge speed in kW. `leak_fraction` defines the loss in the state of charge per hour. `charge_eff` and `discharge_eff` define the efficiency of the charge and discharge process. If not otherwise specified, use the default values.

- `write_output()`:
  Finishes the simulation setup. Must be printed exactly one time in the end.

4 EVALUATION

In this section, two simple examples of small CESs are presented that show the capabilities of GPT and the limitations of the approach. During the writing of the paper, we resorted to a GPT-4 series model. Both examples aim to generate a small CES with a few layers and some atomic components, whereas the load/production curves of consumers and producers should have a typical real-world behavior: a consumer should represent a household with a peak consumption during forenoon and afternoon, while a producer should represent a photovoltaic unit which usually peaks generation during midday. The only differences between the two examples are the number of layers, three for the first and five for the second one, and that the second one adds neighborhood connections, whereas the first one omits these.

The concrete input description and the output from GPT for the first scenario is the following:

Input and answer from GPT for the first scenario

Create a three-layered Cellular Energy System, where each controller has two children. The simulated time should be 24 hours. Randomly add consumers, producers, and storage units to the lowest layer. The consumers should represent households that have a demand peak in the forenoon and the afternoon. The producers should represent a typical photovoltaic unit, e.g., by having a peak at midday and no generation during the night. The storage units have an average capacity of 10 kWh. Do not add neighborhood connections to the system.

- -

Here's a Python program to create the described Cellular Energy System scenario:

```python
import random

def create_consumer_values():
    values = [0] * 288
    for i in range(72, 132):  # Forenoon peak
        values[i] = random.uniform(0.5, 2)
    for i in range(204, 264):  # Afternoon peak
        values[i] = random.uniform(0.5, 2)
    return values

def create_producer_values():
    values = [0] * 288
    for i in range(108, 204):  # Midday peak
        values[i] = random.uniform(0.5, 2)
    return values

def add_energy_cells(controller):
```

```
18    cell_type = random.choice(['consumer', 'producer', 'storage'])
19    if cell_type == 'consumer':
20        add_sub_consumer(controller, create_consumer_values())
21    elif cell_type == 'producer':
22        add_sub_producer(controller, create_producer_values())
23    else:
24        add_sub_storage(controller, capacity=10.0)
25
26 # Generate root controller
27 root = generate_root()
28
29 # Set simulation duration
30 set_simulation_duration(24)
31
32 # Create three-layered Cellular Energy System
33 for _ in range(2):
34     first_layer = add_sub_controller(root)
35     for _ in range(2):
36         second_layer = add_sub_controller(first_layer)
37         for _ in range(2):
38             third_layer = add_sub_controller(second_layer)
39             add_energy_cells(third_layer)
40
41 # Write output
42 write_output()
```

This program creates a three-layered Cellular Energy System with no neighborhood connections. The consumers and producers have demand and generation peaks as described, and the storage units have an average capacity of 10 kWh.

Here, we see, that GPT successfully generated the three layers using three loops. An illustration of the output is given in Figure 3. Furthermore, GPT adds a detailed explanation of what the code does. It is worth noting that depending on the specific execution, GPT may also utilize recursion for the model setup. At the lowest layer, it generates the input curves for the consumers and producers. These are also illustrated in Figure 4. This figure shows that the instructions to model peaks during specific times during the day have been successfully realized, but the curves do not resemble realistic behaviors and only model the peaks as given by the user.
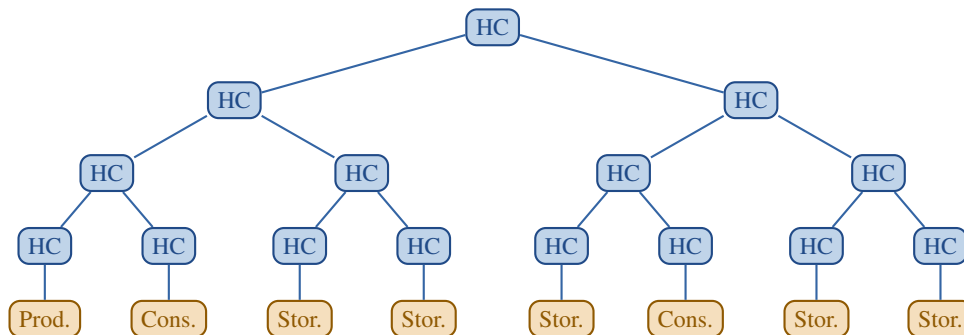

Figure 3: Generated structure from GPT for the first scenario description.

For the second scenario, we provided the following input and received the following output from GPT:
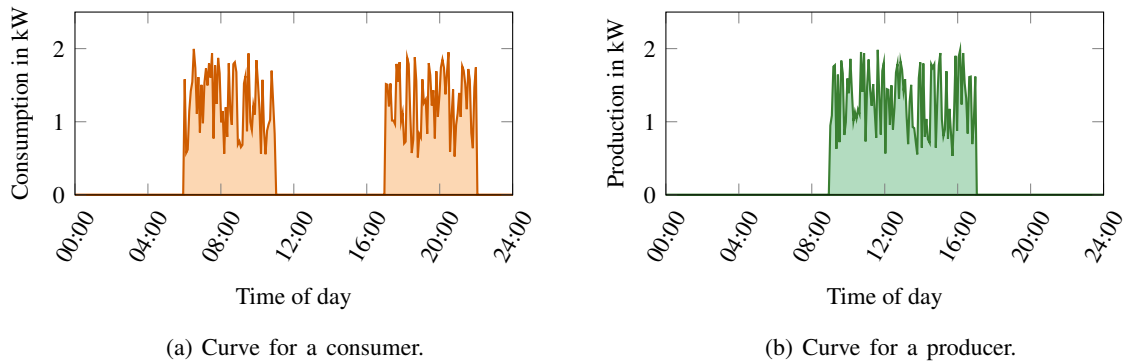
(a) Curve for a consumer.

(b) Curve for a producer.

Figure 4: Generated curves of one consumer and producer from GPT for the first input.

Input and answer from GPT for the second scenario

Create a five-layered Cellular Energy System, where each controller has two children. The simulated time should be 24 hours. Randomly add consumers, producers, and storage units to the lowest layer. The consumers should represent households that have a demand peak in the forenoon and the afternoon. The producers should represent a typical photovoltaic unit, e.g., by having a peak at midday and no generation during the night. The storage units have an average capacity of 10 kWh. Randomly add neighborhood connections between controllers on the same layer.

- -

Here's a Python code snippet for creating the described Cellular Energy System:

```python
import random

def create_consumer_values():
    values = []
    for i in range(288):
        hour = i // 12
        if 7 <= hour <= 10 or 16 <= hour <= 20:
            value = random.uniform(0.8, 1.2)
        else:
            value = random.uniform(0.1, 0.3)
        values.append(value)
    return values

def create_producer_values():
    values = []
    for i in range(288):
        hour = i // 12
        if 9 <= hour <= 15:
            value = random.uniform(2.0, 4.0)
        else:
            value = 0.0
        values.append(value)
    return values

def create_energy_cells(controller, layer):
    if layer == 5:
        cell_type = random.choice(['consumer', 'producer', 'storage'])
        if cell_type == 'consumer':
```

```
29              add_sub_consumer(controller, create_consumer_values())
30          elif cell_type == 'producer':
31              add_sub_producer(controller, create_producer_values())
32          else:
33              add_sub_storage(controller, capacity=10.0)
34      else:
35          sub_controller1 = add_sub_controller(controller)
36          sub_controller2 = add_sub_controller(controller)
37          create_energy_cells(sub_controller1, layer + 1)
38          create_energy_cells(sub_controller2, layer + 1)
39          if random.random() < 0.5:
40              add_neighborhood_connection(sub_controller1, sub_controller2)
41
42  set_simulation_duration(24)
43  root_controller = generate_root()
44  create_energy_cells(root_controller, 1)
45  write_output()
```

(Explanation from GPT omitted)

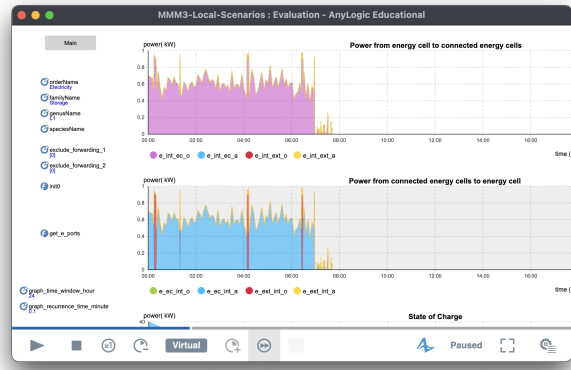Structure file `grid.csv` (representing HCs):

```
1  id;controller_type;output_file
2  grid_0_0;greedy;../output/test/grid_0_0.csv
3  grid_1_0;greedy;../output/test/grid_1_0.csv
4  grid_1_1;greedy;../output/test/grid_1_1.csv
5  [...]
```

[...]

Time series input file `consumer0.csv`:

```
1  year;month;day;hour;minute;value
2  2023;1;1;0;0;0.254202480633273
3  2023;1;1;0;5;0.14071140458586193
4  2023;1;1;0;10;0.2648200686375549
5  [...]
```

[...]

(a) Small selection of input files.

(b) AnyLogic user interface.

Figure 5: Generated input files in the second scenario for i7-AnyEnergy and running simulation in AnyLogic.

For this example, GPT uses recursion which tracks the current recursion depth to find out when the recursion should be terminated. By the given API description, GPT needs to manage the controllers for possible neighborhood connections by itself and cannot resort to fields storing subordinate or neighbor controllers. In this case, this was managed successfully, since in each recursion step the code creates two sub-controllers, and only adds a neighborhood connection between those sub-controllers. However, note that it can happen that GPT falsely assumes at some points that the `controller` data type has the subordinate controllers or other information stored as a field, although the opposite was explicitly stated in the preamble. In such a case, the execution fails. This would require either user adjustments or automatic fallback techniques, e.g., re-running the code generation or removing malicious lines in the code.

Figure 5 demonstrates the next steps that occur in the pipeline. For the given Python API description, we have developed an implementation that creates CSV files for the AnyLogic environment. These CSV files are used for setting up i7-AnyEnergy to spawn the HCs, LCs, and atomic components with the help of AnyLogic *Populations* and connect them afterward. Our implementation automatically finds suitable names for HCs and LCs based on the current layer. As one can see, the representation using CSV files is

well suited to be used in the setup process for the AnyLogic simulation environment, but is less intuitive than the usage of the Python API, showing the need for the intermediate Python representation.

5 CONCLUSION

This study introduces an innovative approach to enhance the end-user experience in complex simulation frameworks through the utilization of an LLM such as OpenAI's GPT. The user does not need to have precise knowledge about the structure of the input files, but can instead specify the desired scenario in a high-level description. We made use of Python scripts as an intermediate layer to facilitate the complexity and size of the output generated from GPT. The evaluation shows that the current version is able to understand the preamble as well as the user input description and create a meaningful scenario configuration script. However, there are limitations to GPT's capabilities. It occasionally produces non-executable codes, particularly as the model complexity increases. Additionally, certain aspects of the system, such as production and generation curves, are rudimentary and lack realistic behavior. The improvement of these issues, either by adjusting the preamble itself or by improving the language model, will be the main focus of future work.

REFERENCES

Anders, B. A. 2023. "Is Using ChatGPT Cheating, Plagiarism, Both, Neither, or Forward Thinking?". *Patterns* 4(3).

Bazan, P. 2017. *Hybrid Simulation of Smart Energy Systems*. Ph. D. thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg.

Dengler, G., P. Bazan, and R. German. 2022, Dec. "Simulation of a Cellular Energy System Including Hierarchies and Neighborhoods". *Energy Informatics* 5(4):51.

Du, H., S. Teng, H. Chen, J. Ma, X. Wang, C. Gou, B. Li, S. Ma, Q. Miao, X. Na, P. Ye, H. Zhang, G. Luo, and F.-Y. Wang. 2023. "Chat with ChatGPT on Intelligent Vehicles: An IEEE TIV Perspective". *IEEE Transactions on Intelligent Vehicles*.

Geerling, W., G. D. Mateer, J. Wooten, and N. Damodaran. 2023. "ChatGPT has Aced the Test of Understanding in College Economics: Now What?". *The American Economist* 68(2):05694345231169654.

Haller, B., O. Langniß, A. Reuter, and N. Spengler. 2020, 12. *1,5° Csellsius - Energiewende zellulär, partizipativ, vielfältig umgesetzt [1.5° Csellsius - Energy Transition Realized Cellularly, Participatively, Diversely]*. C/sells Selbstverlag.

He, X., S. Khan, I. Ozturk, and M. Murshed. 2023, 01. "The Role of Renewable Energy Investment in Tackling Climate Change Concerns: Environmental Policies for Achieving SDG-13". *Sustainable Development* 31:1888–1901.

Jones, D., C. Snider, A. Nassehi, J. Yon, and B. Hicks. 2020. "Characterising the Digital Twin: A Systematic Literature Review". *CIRP Journal of Manufacturing Science and Technology* 29:36–52.

King, M. R., and ChatGPT. 2023. "A Conversation on Artificial Intelligence, Chatbots, and Plagiarism in Higher Education". *Cellular and Molecular Bioengineering* 16(1):1–2.

Kroposki, B., A. Bernstein, J. King, D. Vaidhynathan, X. Zhou, C.-Y. Chang, and E. Dall'Anese. 2020. "Autonomous Energy Grids: Controlling the Future Grid With Large Amounts of Distributed Energy Resources". *IEEE Power and Energy Magazine* 18(6):37–46.

Lalbakhsh, P., V. Mak-Hau, R. Séguin, V. Nguyen, and A. Novak. 2018. "Capacity Analysis for Aircrew Training Schools-Estimating Optimal Manpower Flows under Time Varying Policy and Resource Constraints". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 2285–2296. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Liebrenz, M., R. Schleifer, A. Buadze, D. Bhugra, and A. Smith. 2023. "Generating Scholarly Content with ChatGPT: Ethical Challenges for Medical Publishing". *The Lancet Digital Health* 5(3):e105–e106.

Long, G., X. Binglei, Z. Chuanling, Z. Ning et al. 2023. "Modeling and Simulation Analysis for Urban Rail Transit Hub". *Journal of Civil Engineering and Urban Planning* 5(1):31–42.

Pan, Z., Q. Ou, F. J. Romay, W. Chen, T. You, Y. Liang, J. Wang, and D. Y. Pui. 2023. "Study of Structural Factors of Structure-Resolved Filter Media on the Particle Loading Performance with Microscale Simulation". *Separation and Purification Technology* 304:122317.

Rathore, B. 2023. "Future of Textile: Sustainable Manufacturing & Prediction via ChatGPT". *Eduzone: International Peer Reviewed/Refereed Multidisciplinary Journal* 12(1):52–62.

Retière, N., G. Muratore, G. Kariniotakis, A. Michiorri, P. Frankhauser, J.-G. Caputo, Y. Sidqi, R. Girard, and A. Poirson. 2017. "Fractal Grid–towards the future smart grid". In *CIRED 2017-24th International Conference on Electricity Distribution*. June 12th-15th, Glasgow, United Kingdom, 1236.

Saeed, M. H., W. Fangzong, B. A. Kalwar, and S. Iqbal. 2021. "A Review on Microgrids' Challenges & Perspectives". *IEEE Access* 9:166502–166517.

Song, X., Q. Zhang, Y. Sekimoto, R. Shibasaki, N. J. Yuan, and X. Xie. 2016. "Prediction and Simulation of Human Mobility Following Natural Disasters". *ACM Transactions on Intelligent Systems and Technology* 8(2):1–23.

Tlili, A., B. Shehata, M. A. Adarkwah, A. Bozkurt, D. T. Hickey, R. Huang, and B. Agyemang. 2023. "What if the Devil is my Guardian Angel: ChatGPT as a Case Study of Using Chatbots in Education". *Smart Learning Environments* 10(1):15.

Uhlemeyer, B., M. Becker, J. Garzon-Real, T. Müller, and M. Zdrallek. 2021. "Initial Case Studies Conducted on Cellular Energy Systems at the District Level". In *5th International Hybrid Power Systems Workshop*. May 18th-19th, virtual.

VDE Verband der Elektrotechnik Elektronik Informationstechnik e.V. 2019, May. "Zellulares Energiesystem – Ein Beitrag zur Konkretisierung des zellularen Ansatzes mit Handlungsempfehlung [The Cellular Energy System – A Contribution to the Concretization of the Cellular Approach with a Recommendation for Action]". Technical report, Frankfurt am Main.

Vemprala, S., R. Bonatti, A. Bucker, and A. Kapoor. 2023. "ChatGPT for Robotics: Design Principles and Model Abilities". https://www.microsoft.com/en-us/research/uploads/prod/2023/02/ChatGPT___Robotics.pdf, accessed 20th August 2023.

Wittor, D., and M. Gaspari. 2023. "Unraveling Baroclinicity in Black Hole Weather Storms". *Monthly Notices of the Royal Astronomical Society: Letters* 521(1):L79–L84.

Wolfram, S. 2023. "Wolfram‖Alpha as the Way to Bring Computational Knowledge Superpowers to ChatGPT". https://writings.stephenwolfram.com/2023/01/wolframalpha-as-the-way-to-bring-computational-knowledge-superpowers-to-chatgpt/, accessed 20th August 2023.

Zhang, W., S. Liu, N. Osgood, H. Zhu, Y. Qian, and P. Jia. 2023. "Using Simulation Modelling and Systems Science to Help Contain COVID-19: A Systematic Review". *Systems research and behavioral science* 40(1):207–234.

AUTHOR BIOGRAPHIES

GABRIEL DENGLER is a master's student in Computer Science at Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany, and Research Assistant at the Chair for Computer Networks and Communication Systems at FAU. He holds a bachelor's degree in Computer Science from FAU. Gabriel's research interest lies in modeling and simulation as well as optimization and artificial intelligence in smart energy systems. Furthermore, he focuses on the quantitative analysis of non-markovian systems. His email address is gabriel.dengler@fau.de.

POOIA LALBAKHSH is a Research Assistant at Monash Energy Institute, Faculty of Information Technology, Monash University, Clayton Campus, Australia. Pooia received his Ph.D. in Computer Science and Computer Engineering in 2017 from La Trobe University. He has more than 20 years of experience in academia and industry. Pooia works on digital twins, AI-enabled decision-support systems, Multi-method simulation modeling, and distributed optimization algorithms. His email address is pooia.lalbakhsh@monash.edu.

PETER BAZAN is a Research Assistant at the Chair for Computer Networks and Communication Systems at the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany. He studied Computer Science at FAU and obtained his master's degree (Dipl.-Inf. Univ.) with a thesis on the area of performance modeling. He worked for a couple of years as a consultant for industry projects before he went into business for himself. In addition to his business, he worked for the Performance Modeling and Process Control research group at FAU. Since 2001 he worked in various positions at the Department of Computer Science, where he also received his Ph.D. degree (Dr.-Ing.). He is mainly interested in the modeling and analysis of complex systems. This includes the performance evaluation of computer networks, communication, and production systems, as well as the hybrid simulation of renewable energy systems and smart grids. His email address is peter.bazan@fau.de.

ARIEL LIEBMANN is the Director and co-founder of the Monash Energy Institute and a Professor of Sustainable Energy Systems in the Monash faculty of IT. Ariel's research focus is on the development and application of new decision-support tools for Integrated System Planning for large 100% renewable grids. In 2019 he co-founded the RACE for 2030 CRC as the first Networks Program leader. Ariel is an energy specialist with more than 25 years of experience with 12 years of industry roles in the National Electricity Market. In 2016, Ariel was the architect of the Microgrid/Net Zero Initiative and lead researcher in the Monash Smart Energy City project. His email address is ariel.liebmann@monash.edu.

REINHARD GERMAN is a Full Professor at the Computer Networks Lab in the Department of Computer Science, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), Germany, and an Adjunct Professor at the Faculty of Information Technology of Monash University, Melbourne, Australia. Reinhard received his master's degree in Computer Science in 1991 and his Ph.D. degree in 1994 from the Computer Science Department, Technical University of Berlin, Germany. His research interests include performance and dependability analysis of interconnected systems based on numerical analysis, network calculus, discrete-event simulation, measurements, and testing. Vehicular communications and smart energy constitute major application domains. His email address is reinhard.german@fau.de.