Artificial Intelligence, Social Computing and Wearable Technologies, Vol. 113, 2023, 136–153
https://doi.org/10.54941/ahfe1004187

Prompts of Large Language Model for Commanding Power Grid Operation

Hanjiang Dong[1,2], Jizhong Zhu[1], and Chi-Yung Chung[2]

[1]School of Electric Power Engineering, South China University of Technology, Guangzhou 510641, Guangdong Province, China
[2]Department of Electrical Engineering, The Hong Kong Polytechnic University, Hong Kong

ABSTRACT

Large Language Models (LLMs) like ChatGPT can assist people's general workflows, where the prompt is necessary to inspire the potential of LLMs to solve problems from specified or professional domains like robotics. In the electrical engineering subject or the electric power utility industry, experienced operators and professional experts monitor power grid operation statuses and interact with the grid via human commands on the screen, and components in the grid execute the commands to keep the complex grid safe and economical operation. In this process, human experts edit commands to operate the corresponding software. Human commands are the natural language that the LLM can process. The power grid is composed of generation, transmission, distribution, and other components. Therefore, we redesign the human-computer interaction frame between practitioners and the grid via recurrent prompts to apply the LLM to generate computer programming instructions from the multi-step natural language commands. The programming instruction is executed on system components after being confirmed or revised by human experts, and the quality of generated programs will be gradually improved through human feedback. The idea of this study is originally inspired by studies on controlling individual robotic components by ChatGPT. In the future, we will apply the designed prompt templates to drive the general LLM to generate desired samples which could be used to train an LLM professional in the domain knowledge of electrical engineering to operate multiple types of software for power grid operators.

Keywords: Power grid, Human-computer interaction, Large language model, ChatGPT, Prompt

INTRODUCTION

In the control room of electric power dispatching and control centers, operators are in charge of different tasks to keep the safe and economical operation of the system. Several teams/groups of power grid operators alternate 8-hour or 12-hour shifts to keep the control room staffed around the clock. According to a handbook, *Running the Electric Power Grid*, published by ISO New England in 2016, the control room contains six main operational desks, including the Forecaster, Load Desk, Security Desk, TSO (Tariffs, Schedules, and OASIS) Administrator, Generation Desk, Shift Supervisor, Senior System Operator, and spare Station. Specifically, the Forecaster develops the operating plan given demand curves. The Load Desk responds to real-time conditions and corrects dispatching instructions. The Generation and Security Desks maintain constant readiness by, on the one hand, communicating with power resources (e.g., coal, oil, nuclear, natural gas, solar, wind, and power transfers with other grids) about their capability to execute and, on the other hand, evaluating the effects of possible losses of hundreds of (or thousands of) system components. The TSO Administrator coordinates power transfers with operators in neighboring grid centers to maintain each center on the same page. The Senior System Operator and Shift Supervisor oversee operational activities in the control room to stop abnormal conditions and alert serious issues. Finally, a spare station is staffed in emergencies or utilized to train operators before they are experienced enough to handle real-life situations. At each operational desk, the operator makes the decisions on the power grid operation according to analysis results of running the software for dispatching (Li et al., 2021), forecasting (Dong et al., 2023), and other tasks given the historical records and real-time observations of the power grid states, after coordinating with other desks for accomplishing the operational targets. More detailed investigations for optimal operation (Chen et al., 2021), (Li et al., 2021), (Zhu, 2015) of the power grid (e.g., microgrid, transmission system, and distribution network) and the multi-energy system (micro-energy grid and integrated energy system), as well as energy forecasting of electric power load demand (including electric vehicle charging load) (Dong et al., 2022), (Zhu et al., 2022), the number of vehicle cars, including electric vehicles (Dong et al., 2020), renewable power generation, such as photovoltaic and wind, have made related software tend to be mature in open-source and commercial fields. One of the key future trends is how to select, combine and implement mature functional software properly to accomplish the power grid operating target.

In Natural Language Processing, a *large language model (LLM)* is defined as a sophisticated computational system designed to generate human-like text based on input prompts[1]. The LLM is typically developed based on deep learning techniques, specifically leveraging architectures such as recurrent neural networks (RNNs) and transformers. OpenAI's GPT-3 has recently attracted the attention of the globe and all walks of life and has become a prime example of an LLM. LLMs are trained on massive amounts of text or language data from diverse sources, e.g., books, articles, websites, and many other textual resources. The training process involves exposing the LLM to vast quantities of text-related data, allowing it to learn adaptive patterns, abstract relationships, and semantic representations of language (Zhao, 2023). The underlying architecture of the LMM enables it to understand and generate coherent and contextually relevant responses that are also known as human-like text and can span multiple paragraphs or even much longer compositions to given *prompts*. Due to its size and complexity derived from the learned patterns and associations within the training data, the LLM possesses an expansive knowledge base, opening access to an

[1] Output from ChatGPT. Open AI. Prompt: "Give me the academic definition of the large language model." 11 July 2023.

extraordinarily wide range from factual information to linguistic structures to semantic relationships (Liu et al., 2021).

Lately, LLMs, especially recent advances of GPT-3 and GPT-4 (OpenAI, 2022), have inspired valuable tools for numerous applications, including natural language understanding, text generation, translation, summarization, and even assisting with creative writing tasks in some general fields. In other words, the trained LLMs are generally suitable for general tasks and can thus be fine-tuned for specific tasks in some professional domains, such as robotics (Wake, 2023). For instance, the LLMs could be applied to convert natural language commands of humans into executable computer programming instructions with natural language explanations through a readable JSON format while representing changeable operating environments in a formalized style and updating them as one of the next inputs together with the next command to take advantage of the memory of the latest manipulations. Before fine-tuning the general LLM into the professional LLM, prompts are designed to obtain the samples from the general LLM, and some capabilities of a general LLM can be stimulated to make it serve as the required character through the input prompts. As the development of the LLM is still in its early rising period, there is no established system or methodology yet. The prompts should be easy-to-customize for the LLM and meet common requirements in the practical application of the professional area (Wake, 2023), for instance, (1) easy integration with existing systems, software, or tools, (2) applicability to varying environments, (3) the ability to provide long-step commands while minimizing the impact of the LLM's token limit.

Just like robotics, generating programs for executing commands of operating the multiple components for transmission, distribution, and generation of electricity is a new promising goal considering the power grid is constructed and interconnected with individual components. Meanwhile, the algorithms or methods for forecasting, dispatching, and controlling the power grid are maturing. With the help of the LLM, more attention can be paid to combining or implementing these functional methods appropriately. However, the works that design prompt templates for harnessing the language and interaction capabilities of the LLM as a professional interface with a small number of samples to tune up the LLM for arranging mature components or software in specific fields are few, and there is no practical application in electrical engineering (See Figure 1).

To fill the gap, we design the prompt templates that enable the LLM to transform natural language commands into computer programming instructions, assisting the system operator or practitioner in summarizing power grid states and executing the operational commands recurrently through the functional software tedious to input instructions after being confirmed or corrected by human experts. Hence, operators may focus more on the system operation situations and further improve operational services and strategies. In electrical engineering, this work is the first one exploring the prompt templates for the LLM, based on the study of prompts in robotics. More investigations are imperative to leverage the power of the LLM.

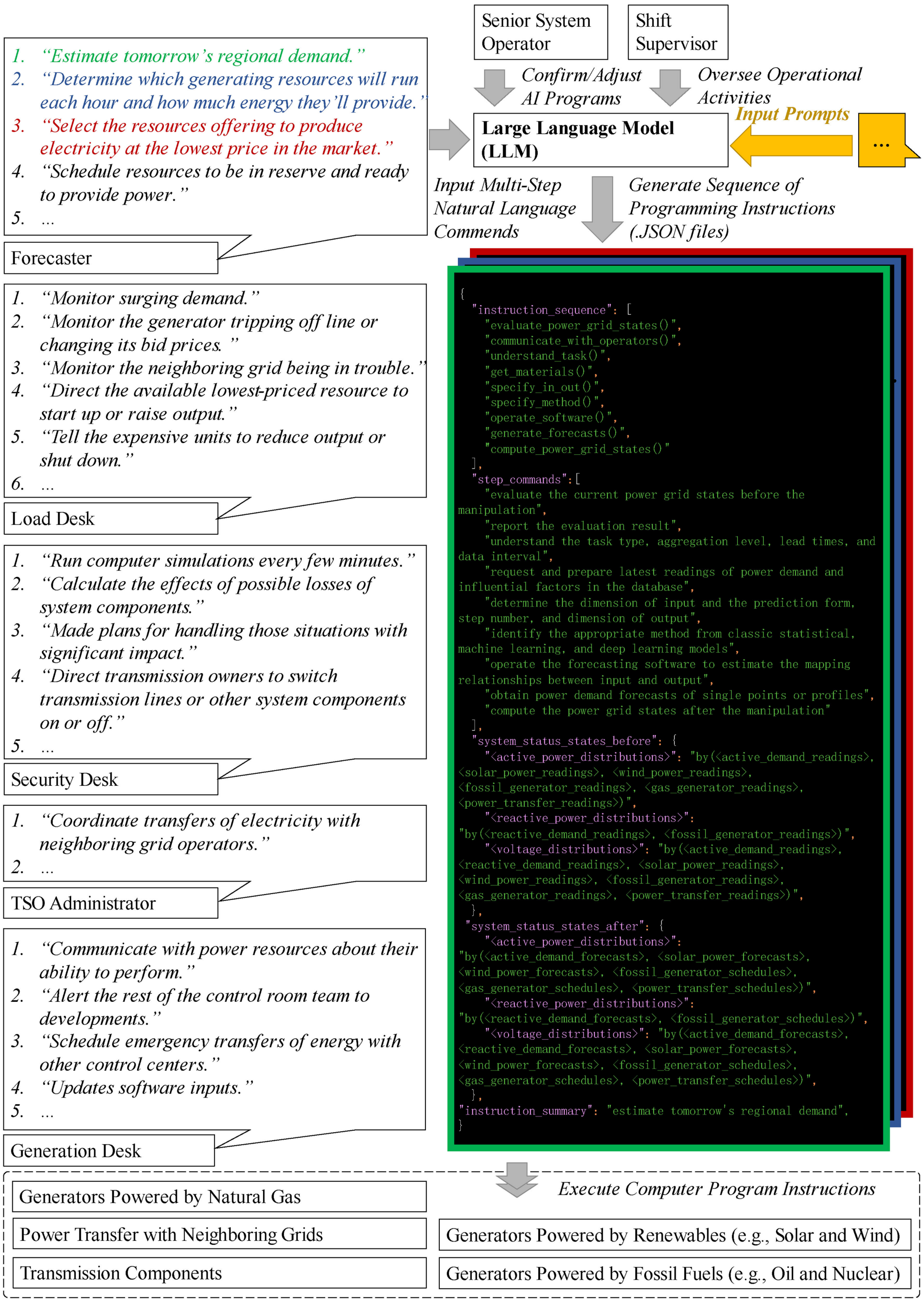

Figure 1: Prompt templates for large language model to generate a sequence of computer programming instructions from multi-step natural language commands coordinating various components of generation, transmission, etc., and power transfer with neighboring grids as a power grid assistant operator in diverse operational environments. (Adapted from the ISO New England, running the electric power grid, 2016).

PROMPT TEMPLATES FOR LARGE LANGUAGE MODEL

In this section, we mainly design six main prompts for driving the LLM to generate executable computer programming instructions given natural language commands of human experts by modifying the prompts designed for controlling robots (Wake, 2023), including (1) the clarification of the role of the LLM as an assistant operator, (2) the definition of system operational commands, (3) the representation of power grid operational environments, (4) the explanation of the output format of the LLM, (5) three examples of the input and output of the LLM, and (6) specified input from the system operator.

At a specific operational workflow, the operator inputs the six prompts above as a template group, together with a natural language command; the LLM outputs the computer programming instructions, containing the evaluation of power grid states or statuses before and after the exact part corresponding to the input command, and the power grid states can be recorded as the environmental information and utilized as one of the next inputs recurrently; the feedbacks from human experts during the workflow are also allowed to fine-tune the LLM and improve the performance after the first workflow according to the real-time utilization experience (See Figure 2).

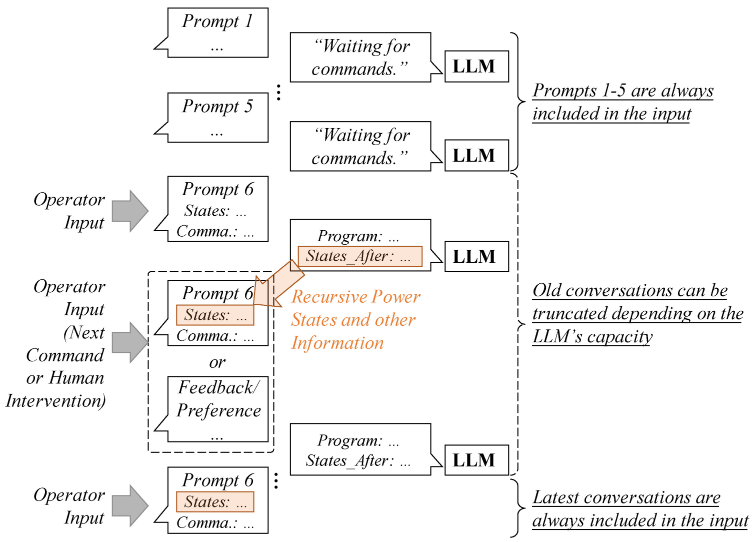

Figure 2: The whole process of the conversation between the LLM and the system operator.

The Role of Large Language Model as an Assistant System Operator

The first prompt provides the context for this specific task of generating executable computer programming instructions from natural language commands and clarifies the role that the LLM should play as an assistant operator. In addition, an additional sentence is also input at the end of the prompt to start working until the six prompts are input to the LLM (See Figure 3).

```
You are an excellent interpreter of natural language commands of power grid operators during the power grid
operation.
Given a command from operators at the six key desks in the control room, including the Forecaster, Load Desk,
Security Desk, TSO (Tariffs, Schedules, and OASIS) Administrator, Generation Desk, and Senior System Operator,
and information about the operation states of the power grid, you break it down into multi-step natural
language commands and generate a sequence of computer programming instructions executed on the components of
the grid, including generation, distribution, and transmission, after being confirmed or adjusted by the
Senior System Operator.
Please do not begin working until I say, "Start working." Instead, simply output the message "Waiting for next
input."
Understand?
```

Figure 3: Prompt 1 for explaining the role of the LLM.

Definition of Power Grid Operational Commands

The second prompt defines and explains the set of power grid operational computer programming instructions in the manner of callable functions or packaged modules, including consistent processes of evaluating power grid states, understanding tasks, getting related materials, specifying the required input and output result, specifying the core method with the specified input, operating related software of the specified method, generating energy prediction results if the software is about the forecasting task, communicating with power resources for the ability to perform the scheduling results if the software is of the dispatching task, specifying the confirmed schedules, computing system states after executing confirmed schedules, and communicating with system operators to ensure the system states before and after the command. It should be noted that the processes or steps mentioned above are general procedures of the operator's regular workflows, and each of them involves varying content of forecasting, dispatching, and other operational tasks introduced in the Introduction. Similarly, an extra sentence is supplemented at the end to instruct the LLM to wait for the next prompt until all the prompts are input (See Figure 4).

Representation of Power Grid Operational Environments

The third prompt defines the method for representing the related information of the power grid operational environments. The environment is divided into *components* of the power grid, such as fossil (coal, oil, and nuclear) generators, natural gas units, power transfer lines, solar, e.g., photovoltaic, power generation panels, wind power generation turbines, active power demand loads, and reactive power compensators/ loads, and *system statuses* of the power grid, including reactive power distributions, active power distributions, and voltage distributions. It shall be noted that we adopt the word *status* to represent the power grid states instead of "state" to represent the values of the power grid states to avoid possible misunderstandings or overlaps of *component states* and *system status states*. Besides, every component involves two major halves, i.e., the historical half of the "here-and-now" and known information, such as *wind_ power_readings*, and the result half of the "wait-and-see" and output /generated target, e.g., *wind_power_forecasts*. Then, every output of the result half can be obtained by inputting the historical half to the selected software through the actions defined in the *"STATE*

```
Necessary and sufficient power grid operational computer programming instructions are defined as follows:
"""
"COMPUTER PROGRAMMING INSTRUCTION LIST"
- understand_task(): Understand the task type, conditions, and requirements.
- get_materials(): Request and prepare related materials, such as power grid layout, components and their
parameters, and historical data in the database. get_materials() is allowed only when understand_task() is
finished.
- specify_in_out(): Determine the necessary input parameters and the required output results of the computer
programming instruction. specify_in_out() is allowed only when get_materials() is finished.
- specify_method(): Determine the appropriate key technical method of the computer programming instruction
from candidate methods. specify_method() is allowed only when specify_in_out() is finished. For instance, the
candidate forecasting models from a tested set of classic statistical, machine learning, and deep learning
models. Furthermore, transfer learning, multi-task learning, and other strategies could improve the candidate
forecasting models.
- operate_software(): Run the related software to realize the specified method. operate_software() is allowed
only when specify_method() is finished. For instance, parameter assignments of the forecasting model mapping
the relationship between influential factors, such as weather indexes, power prices, and calendar information,
and target power demand are optimized with their readings. The parameters differ in different forecasting
models.
For another instance, based on the power grid operation model, optimal decision variable values of the
optimization model in economic dispatch are optimized with assignments of the power grid, component, and
optimization model parameters.
- generate_forecasts(): Obtain single-point or profile energy forecasts, such as power load demand,
photovoltaic power generation, and wind power generation. generate_forecasts() is allowed only when
operate_software() is finished.
- communicate_with_power_resources(): Communicate with the power resources about their ability to perform,
including generators or panels powered by coal, oil, nuclear resources, natural gas, and transfers of
electricity with other control centers of neighboring grids. communicate_with_power_resources() is only
allowed when operate_software() is finished.
- communicate_with_operators(): Alert the team of system operators about the abnormal situations of the power
grid states. communicate_with_operators() is only allowed when compute_power_grid_states() is finished.
- specify_schedules(): Specify the on/off/sold/buy commitments of power resources and the power
outputs/transactions of those being started up or traded. specify_schedules() is only allowed as the
confirmation or adjustment of the outcomes from the finished operate_software(), only after
communicate_with_power_resources() is finished.
- compute_power_grid_states(): Compute the power grid states of active power flow, reactive power flow, and
voltage distributions among the buses connected with transmission lines in the system.
compute_power_grid_states() is only allowed when specify_schedules() is finished, given the system layout and
parameters of the power grid and its components.
- evaluate_power_grid_states(): Compare the actual status of the power grid and power resources with the
limits, referring to the alerts of abnormal situations from monitoring software. evaluate_power_grid_states()
is only allowed when specify_schedules() is finished.
"""
-----------------------------------------------------------------
The texts above are part of the overall command. Do not start working yet:
```

Figure 4: Prompt 2 for explaining the power grid operational functions.

LIST," such as *by (<something>)*. An extra sentence is also added at the end to instruct the LLM to wait for the next prompt until all the prompts are input (See Figure 5).

```
Information about environments and power grid states is given as a Python dictionary. Example:
"""
{"environment": {"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "by(<active_power_demand_readings>,
<influential_factors>)",
  "<reactive_power_demand_forecasts>": "by(<reactive_power_demand_readings>, <influential_factors>)",
  "<solar_generator_forecasts>": "by(<solar_readings>)",
  "<wind_generator_forecasts>": "by(<wind_readings>)",
  "<fossil_generator_schedules>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_forecasts>, <wind_generator_forecasts>)",
  "<gas_generator_schedules>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_forecasts>, <wind_generator_forecasts>)",
  "<power_transfer_schedules>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_forecasts>, <wind_generator_forecasts>)"},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_forecasts>,
<solar_generator_forecasts>, <wind_generator_forecasts>, <fossil_generator_schedules>,
<gas_generator_schedules>, <power_transfer_schedules>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_forecasts>, <fossil_generator_schedules>)",
  "<voltage_distributions>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_forecasts>, <wind_generator_forecasts>, <fossil_generator_schedules>,
<gas_generator_schedules>, <power_transfer_schedules>)"}}}"""
"STATE LIST"
- by(<something>): Power grid state is obtained with <something>
- same_as(<something>): Power grid state is the same as <something>
"""
<something> should be one of the components or power grid states in the environment.
-----------------------------------------------------------------
The texts above are part of the overall command. Do not start working yet:
```

Figure 5: Prompt 3 for the representation of the power grid operational desks.

The Format of the Output From the Large Language Model

A Python dictionary, which can be saved as a JSON file to integrate other programs or processes, is explained as the output format of the LLM within the fourth prompt. The dictionary involves six crucial aspects: task cohesion, power grid states before, power grid states after, command summary, and abnormal situation report. Besides, the task cohesion mainly contains the instruction sequence, the system status name, and step commands (See Figure 6). A natural language command input to the LLM will then be divided into multi-step commands to build the dictionary with detailed descriptions through the LLM. An extra sentence is added at the end to instruct the LLM to wait for the next prompt until all the prompts are input.

```
You divide the commands in the text into detailed natural language commands and combine them as a Python
dictionary.
The dictionary has six keys.
"""
- dictionary["task_cohesion"]: A dictionary containing information about the power grid operational commands
that have been split up.
- dictionary["power_grid_states_before"]: The state of the power grid status before the manipulation.
- dictionary["power_grid_states_after"]: The state of the power grid status after the manipulation.
- dictionary["command_summary"]: The brief summary of the given sentence or text.
- dictionary["question"]: If you cannot understand the given sentence or text, you can ask the team of system
operators to rephrase the sentence. Leave this key empty if you can understand the given sentence or text.
- dictionary["abnormal_situation_report"]: If you find any abnormal situation after receiving the power grid
states, you can ask the team of system operators to check the data in workflows. Leave this key empty if the
power grid works as commanded.
"""
Three keys exist in the dictionary["task_cohesion"].
- dictionary["task_cohesion"]["instruction_sequence"]: Contains a list of computer programming instructions.
Only the functions defined in the "COMPUTER PROGRAMMING INSTRUCTION LIST" will be used.
- dictionary["task_cohesion"]["step_commands"]: Contains a list of natural language commands for the power
grid corresponding to the list of computer programming instructions.
- dictionary["task_cohesion"]["system_status_name"]: The name of the manipulated power state. Only power grid
states defined in the input dictionary will be used for the power grid state name.
"""
<something> should be one of the components or grid parameters in the operational desk.
-------------------------------------------------------------------
The texts above are part of the overall command. Do not start working yet:
```

Figure 6: Prompt 4 for explaining the format of the operational commands generated by the language model.

Examples of the Input and Output of the Language Model

Prompt 5 provides three examples of the expected inputs and outputs related to the LLM, and the three commands in green, blue, and red fonts in Figure 1 are included in the three inputs, respectively (See Figure 7). Extensive examples corresponding to the commands exhibited in Figure 1 are welcome, which is hard to obtain. Hence, we only consider three examples to fine-tune the LLM to obtain effective few-shot results in this study (Brown et al., 2020). Similarly, we add an extra sentence at the end of this prompt to instruct the LLM to wait for the final prompt.

````
I will give you examples of the input that you will receive and the output that you will generate.
Example 1:
"""
- Input:
{"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "",
  "<reactive_power_demand_forecasts>": "",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "",
  "<gas_generator_schedules>": "",
  "<power_transfer_schedules>": ""},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_readings>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_readings>, <fossil_generator_readings>)",
  "<voltage_distributions>": "by(<active_power_demand_readings>, <reactive_power_demand_readings>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)"}
"command": "Estimate tomorrow's regional demand"}
- Output
```
{"task_cohesion": {
  "instruction_sequence": [
    "evaluate_power_grid_states()",
    "communicate_with_operators()",
    "understand_task()",
    "get_materials()",
    "specify_in_out()",
    "specify_method()",
    "operate_software()",
    "generate_forecasts()",
    "compute_power_grid_states()"
  ],
  "step_commands": [
    "evaluate the current power grid states before the manipulation",
    "report the evaluation result",
    "understand the task type, aggregation level, lead times, and data interval",
    "request and prepare latest readings of power demand and influential factors in the database",
    "determine the dimension of input and the prediction form, step number, and dimension of output",
    "identify the appropriate method from classic statistical, machine learning, and deep learning models",
    "operate the forecasting software to estimate the mapping relationships between input and output",
    "obtain power demand forecasts of single points or profiles",
    "compute the power grid states after the manipulation"
  ],
"system_status_name": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"]},
"environment_before": {"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "",
  "<reactive_power_demand_forecasts>": "",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "",
  "<gas_generator_schedules>": "",
  "<power_transfer_schedules>": ""},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_readings>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_readings>, <fossil_generator_readings>)",
  "<voltage_distributions>": "by(<active_power_demand_readings>, <reactive_power_demand_readings>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)"}},
"environment_after": {"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
````
````

~~~~
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "by(<active_power_demand_readings>,
<influential_factors>)",
  "<reactive_power_demand_forecasts>": "by(<reactive_power_demand_readings>, <influential_factors>)",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "",
  "<gas_generator_schedules>": "",
  "<power_transfer_schedules>": ""},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_forecasts>, <fossil_generator_readings>)",
  "<voltage_distributions>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)"}},
"command_summary": "estimate tomorrow's regional demand",
"question": "",
"abnormal_situation_report": ""}
```
"""
Example 2:
"""
- Input:
{"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "by(<active_power_demand_readings>,
<influential_factors>)",
  "<reactive_power_demand_forecasts>": "by(<reactive_power_demand_readings>, <influential_factors>)",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "",
  "<gas_generator_schedules>": "",
  "<power_transfer_schedules>": ""},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_forecasts>, <fossil_generator_readings>)",
  "<voltage_distributions>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)"}
"command": "Determine which generating resources will run each hour and how much energy they'll provide"}
- Output
```
{"task_cohesion": {
  "instruction_sequence": [
    "evaluate_power_grid_states()",
    "communicate_with_operators()",
    "understand_task()",
    "get_materials()",
    "specify_in_out()",
    "specify_method()",
    "operate_software()",
    "communicate_with_power_resources()",
    "specify_schedules()",
    "compute_power_grid_states()"
  ],
  "step_commands": [
    "evaluate the current power grid states before the manipulation",
    "report the evaluation result",
    "understand the power grid model, optimization objectives, constraints, lead times, and interval",
    "request and prepare latest readings of optimization model parameters, such as power demand, in the
database",
    "determine parameter assignments and define decision variables",
    "identify the appropriate method from linear programming, non-linear programming, and mix-integer
programming",
    "operate the dispatching software to obtain the optimal solutions of decision variables",
    "communicate with power resources about their ability to execute the decision solutions",
    "determine the final schedules according to the feedbacks from power resources",
    "compute the power grid states after the manipulation"
  ],
"system_status_name": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"]},
"environment_before": {"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "by(<active_power_demand_readings>,
<influential_factors>)",
  "<reactive_power_demand_forecasts>": "by(<reactive_power_demand_readings>, <influential_factors>)",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "",
  "<gas_generator_schedules>": "",
~~~~

````
  "<power_transfer_schedules>": ""},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_forecasts>, <fossil_generator_readings>)",
  "<voltage_distributions>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)"}},
"environment_after": {"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "by(<active_power_demand_readings>,
<influential_factors>)",
  "<reactive_power_demand_forecasts>": "by(<reactive_power_demand_readings>, <influential_factors>)",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>)",
  "<gas_generator_schedules>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>)",
  "<power_transfer_schedules>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>)"},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_schedules>,
<gas_generator_schedules>, <power_transfer_schedules>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_forecasts>, <fossil_generator_schedules>)",
  "<voltage_distributions>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_schedules>,
<gas_generator_schedules>, <power_transfer_schedules>)"}},
"command_summary": "determine which generating resources will run each hour and how much energy they'll
provide",
"question": "",
"abnormal_situation_report": ""}
```
"""
Example 3:
"""
- Input:
{"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "by(<active_power_demand_readings>,
<influential_factors>)",
  "<reactive_power_demand_forecasts>": "by(<reactive_power_demand_readings>, <influential_factors>)",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "",
  "<gas_generator_schedules>": "",
  "<power_transfer_schedules>": ""},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_forecasts>, <fossil_generator_readings>)",
  "<voltage_distributions>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)"}
"command": "Select the resources offering to produce electricity at the lowest price in the market"}
- Output
```
{"task_cohesion": {
  "instruction_sequence": [
    "evaluate_power_grid_states()",
    "communicate_with_operators()",
    "understand_task()",
    "get_materials()",
    "specify_in_out()",
    "specify_method()",
    "operate_software()",
    "communicate_with_power_resources()",
    "specify_schedules()",
    "compute_power_grid_states()"
  ],
  "step_commands": [
    "evaluate the current power grid states before the manipulation",
    "report the evaluation result",
    "understand the power grid model, optimization objectives, constraints, lead times, and interval",
    "request and prepare latest readings of optimization model parameters, such as power demand and
electricity price, in the database",
    "determine parameter assignments and define decision variables",
    "identify the appropriate method from linear programming, non-linear programming, and mix-integer
programming",
    "operate the dispatching software to obtain the optimal solutions of decision variables at the lowest
price",
    "communicate with power resources about their ability to execute the decision solutions",
    "determine the final schedules according to the feedbacks from power resources",
    "compute the power grid states after the manipulation"
  ],
````

````
"system_status_name": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"]},
"environment_before": {"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "by(<active_power_demand_readings>,
<influential_factors>)",
  "<reactive_power_demand_forecasts>": "by(<reactive_power_demand_readings>, <influential_factors>)",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "",
  "<gas_generator_schedules>": "",
  "<power_transfer_schedules>": ""},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_forecasts>, <fossil_generator_readings>)",
  "<voltage_distributions>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)"}},
"environment_after": {"components": ["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "by(<active_power_demand_readings>,
<influential_factors>)",
  "<reactive_power_demand_forecasts>": "by(<reactive_power_demand_readings>, <influential_factors>)",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>)",
  "<gas_generator_schedules>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>)",
  "<power_transfer_schedules>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>)"},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_schedules>,
<gas_generator_schedules>, <power_transfer_schedules>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_forecasts>, <fossil_generator_schedules>)",
  "<voltage_distributions>": "by(<active_power_demand_forecasts>, <reactive_power_demand_forecasts>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_schedules>,
<gas_generator_schedules>, <power_transfer_schedules>)"}},
"command_summary": "select the resources offering to produce electricity at the lowest price in the market",
"question": "",
"abnormal_situation_report": ""}
```
"""
From these examples, learn that some computer programming instructions have dependencies with the instructions
before and after them.
-------------------------------------------------------
The texts above are part of the overall command. Do not start working yet:
````

Figure 7: Prompt 5 for system operators providing examples of the desired input and output of the language model. A part of the prompt is shown.

Specified Input From the Power Grid Operator

The final prompt formats the input specified by the system operator in terms of the environmental information and natural language command. The *[ENVIRONMENT]* and *[COMMAND]* can be assigned or replaced with the next input after processing the first command when the *[ENVIRONMENT]* is blank. Hence, the environmental information at the last time is taken advantage of at the next recurrence (See Figure 8). Furthermore, Prompt 6 defines special rules on the generated Python dictionary for the LLM to follow. Now, the LLM can start generating computer programming instructions after the operator inputs the six essential prompts, including the natural language command, and coordinate the components in the power grid by executing the generated programs after being confirmed or corrected by the experts. It should be noted that the LLM only generates JSON programs that operate existing systems, tools, or software automatically in this study instead of the academic programs that design the algorithms or methods for
````

dispatching, forecasting, and other tasks with real-time observations or readings of power grid states, which is extremely difficult to accomplish in the current stage. In other words, in our opinion, designed prompts will only reliably do as commanded and timely alert abnormal situations following the standard operational workflow while saving time from tedious works generally completed by humans without the creativity of academic or engineering content.

````
Start working. Resume from the environment below.
"""
{"environment": [ENVIRONMENT]}
"""
The command is as follows:
"""
{"command": [COMMAND]}
"""
The dictionary that you return should be formatted as a Python dictionary. Follow these rules:
1. The first element should be understand_task() to specify task type, conditions, and requirements to
evaluate power grid states.
2. Make sure that each element of the ["step_commands"] explains the corresponding element of the
["instruction_sequence"]. Refer to the "COMPUTER PROGRAMMING INSTRUCTION LIST" to understand the elements of
["instruction_sequence"].
3, The length of the ["step_ commands"] list must be the same as the length of the ["task_sequence"] list.
4. Even though the power grid states disappear, do not delete them from the input and use the "STATE LIST" to
keep track of all the power grid states.
5. Ensure you output a consistent or continuous manipulation as a single process. For example, getting
materials or inputting the data should not occur in successive steps.
6. Never left ',' at the end of the list.
7. All keys of the dictionary should be double-quoted.
8. Insert ``` at the beginning and the end of the dictionary to separate it from the rest of your response.
Adhere to the output format I defined above. Think step by step, being aware of what the operational desk is
responsible for.
````

(a): Input at the last part of the input

```
environment: {"components":["<fossil_generator_readings>", "<fossil_generator_schedules>",
  "<gas_generator_readings>", "<gas_generator_schedules>",
  "<power_transfer_readings>", "<power_transfer_schedules>",
  "<solar_readings>", "<solar_generator_forecasts>", "<wind_readings>", "<wind_generator_forecasts>",
  "<active_power_demand_readings>", "<active_power_demand_forecasts>",
  "<reactive_power_demand_readings>", "<reactive_power_demand_forecasts>", "<influential_factors>"],
"component_states": {"<active_power_demand_forecasts>": "",
  "<reactive_power_demand_forecasts>": "",
  "<solar_generator_forecasts>": "",
  "<wind_generator_forecasts>": "",
  "<fossil_generator_schedules>": "",
  "<gas_generator_schedules>": "",
  "<power_transfer_schedules>": ""},
"system_statuses": ["<active_power_distributions>", "<reactive_power_distributions>",
"<voltage_distributions>"],
"system_status_states": {"<active_power_distributions>": "by(<active_power_demand_readings>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)",
  "<reactive_power_distributions>": "by(<reactive_power_demand_readings>, <fossil_generator_readings>)",
  "<voltage_distributions>": "by(<active_power_demand_readings>, <reactive_power_demand_readings>,
<solar_generator_readings>, <wind_generator_readings>, <fossil_generator_readings>, <gas_generator_readings>,
<power_transfer_readings>)"}
}
command: "Estimate tomorrow's regional demand"
```

(b): Example of an environment and command

Figure 8: Prompt 6 for system operators inputting template and examples of the actual input. Continuous power grid operations are realized by defining the environment information after the operational output by the LLM as the next input.

IMPLEMENTATION SCHEME AND CASE STUDY

To validate the feasibility of the prompt scheme above, we design a case study and implement this scheme, including (1) multi-step manipulations of natural language commands, and (2) correction of the output by feedback from operators. The case study is conducted with one of the most dominant LLMs, i.e., ChatGPT based on GPT-3.5 turbo (OpenAI, 2022), which can be easily accessed by visiting the website or an open-source API on personal computers.

Multi-Step Manipulations of Natural Language Commands

Based on the input prompts, ChatGPT outputs JSON files, a part of which has been shown in Figure 9, of a sequence of computer programming instructions from step-by-step natural language commands given the environmental information of power grid states. The example from Figure 9 involves four commands in Forecaster (See Figure 1, including:

(a) *”Estimate tomorrow’s regional demand.”*
(b) *”Determine which generating resources will run each hour and how much energy they’ll provide.”*
(c) *”Select the resources offering to produce electricity at the lowest price in the market.”*
(d) *”Schedule resources to be in reserve and ready to provide power.”*

Figure 9: Multi-step manipulations of a sequence of computer programming instructions for step-by-step natural language commands to a chosen LLM, ChatGPT, given environmental information.

The sequences of functions or modules in JSON programs for human commands are similar because the designed prompts require the selected ChatGPT to generate programs following standard operational workflows of the power grid. Meanwhile, each natural language command input to ChatGPT has been broken down into step-by-step commands by the LLM, illustrating that similar sequences of programming functions or modules involve varying specific contents (tools, software, or systems) for different tasks. Through the LLM, one of the most important jobs for the system operator is to package and improve the modules while confirming or correcting the generated instructions and step-by-step commands instead of realizing the required manipulations manually, which may be easy to make mistakes.

Moreover, the GPT-3.5 turbo exhibits the environmental information before and after each manipulation. The latest component states after the last manipulation are adopted to compute the next *system statuses* in terms of reactive power distribution, active power distribution, and voltage distribution (See Figure 10).

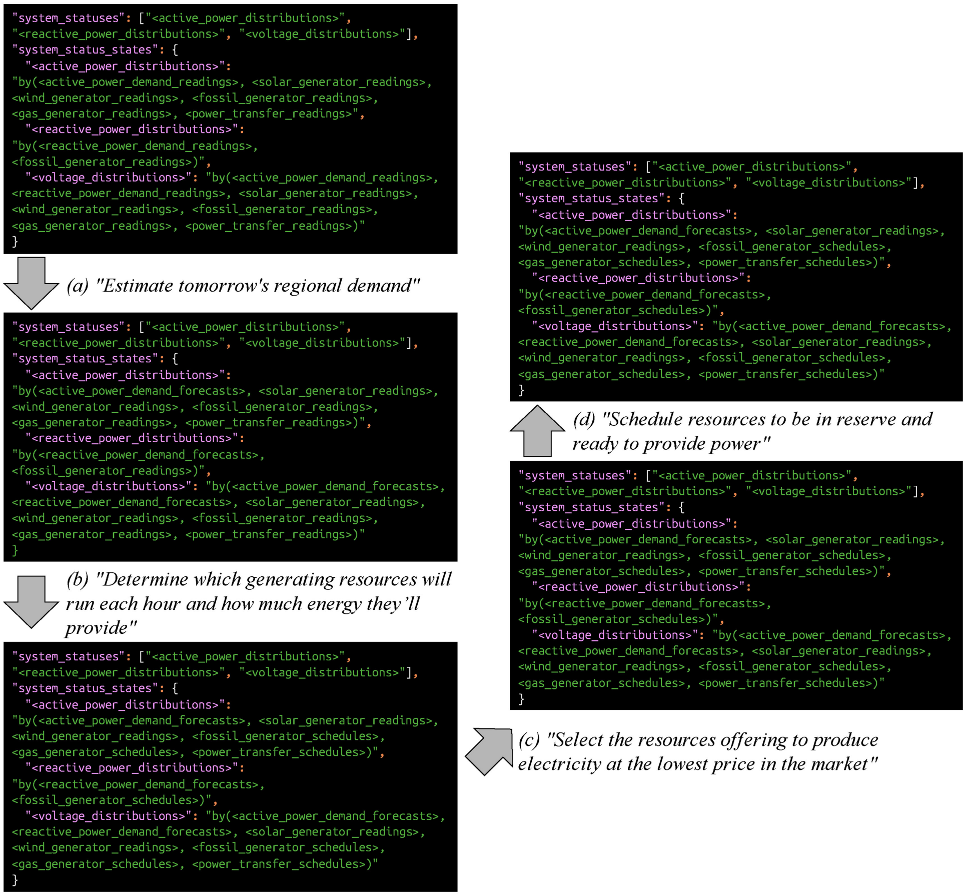

Figure 10: The environmental information output by the chosen LLM called ChatGPT during multi-step manipulations.

Correction of the Output via Feedback from Operators

With the real-time conversational design characteristic of ChatGPT, the power grid operator can check and correct the output for each natural language command with domain knowledge in real-life situations. In other words, human interventions such as feedback and preference are allowed to correct the following output, for instance, inserting more or deleting specific functions or modules in the programs. The LLM then outputs the revised program again until the operator confirms to execute it and inputs the next natural language command (See Figure 11). The example shown in Figure 11 involves the specified command "Determine which generating resources will run each hour and how much energy they'll provide" with human Intervention. We believe correcting the output will be the key to improving the performance.

Figure 11: Example of correcting the sequence of instructions for the command "determine which generating resources will run each hour and how much energy they'll provide" via the LLM, ChatGPT, with human Intervention (feedback or preference).

CONCLUSION

In summary, this work designed the prompt templates to apply the LLM to generate sequences of computer programming instructions from multistep natural language commands while considering the latest environmental information, e.g., the power grid operational states. The first application of the LLM for commanding the power grid for the system operator would change the human-computer interaction manner between operators and the power grid. The designed prompts could be utilized and tested with different LLMs besides the GPT models in the future. Future work will also realize higher-level logic of the LLM, for instance, generating the program for the conditional command *"Set regional demand readings as the estimated value if the out-of-date day is expired. Otherwise, estimate tomorrow's regional demand."*

ACKNOWLEDGMENT

The authors would like to acknowledge Prof. Dong Zhaoyang (School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore) for his constructive advice for developing LLM-based human-computer interactions. This study was in part supported by the High-end Foreign Experts Project (grant number G2022163018L).

REFERENCES

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, P., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. & Amodei, D. (2020). Language Models are Few-Shot Learners. arXiv:2005.14165. DOI: https://doi.org/10.48550/arXiv.2005.14165

Chen, Z., Zhu, J., Dong, H., Wu, W. & Zhu, H. (2021). Optimal dispatch of WT/PV/ES combined generation system based on cyber-physical-social integration. IEEE Transactions on Smart Grid, 13(1), 342–354.

Dong, H., Zhu, J., Li, S. & Chen, Z. (2020). Model-driven forecasting for the number of private electric vehicles considering social propaganda and subsidy policy. Proceedings of 12 the International Conference on Applied Energy, 10, 1–4.

Dong, H., Zhu, J., Li, S., Luo, T., Li, H. & Huang, Y. (2022). Ultra-short-term load forecasting based on convolutional-LSTM hybrid networks. 2022 IEEE 31st International Symposium on Industrial Electronics, 2022, 142–148.

Dong, H., Zhu, J., Li, S., Wu, W., Zhu, H. & Fan, J. (2023). Short-term residential household reactive power forecasting considering active power demand via deep Transformer sequence-to-sequence networks. Applied Energy, 329, 120281.

Li, S., Zhu, J. & Dong, H. (2021). A novel energy sharing mechanism for smart microgrid. IEEE Transactions on Smart Grid, 12(6), 5475–5478.

Li, S., Zhu, J., Dong, H., Zhu, H. & Fan, J. (2022). A novel rolling optimization strategy considering grid-connected power fluctuations smoothing for renewable energy microgrids. Applied Energy, 309, 118441.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H. & Neubig G. (2021). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. arXiv: 2107.13586. DOI: https://doi.org/10.48550/arXiv.2107.13586

OpenAI. (2022). ChatGPT Website: https://openai.com/blog/chatgpt

Wake, N., Kanehira, A., Sasabuchi, K., Takamatsu, & J., Ikeuchi, K. (2023). ChatGPT empowered long-step robot control in various environments: A case application. arXiv: 2304.03893. DOI: https://doi.org/10.48550/arXiv.2304.03893

Zhao, W. X., Zhou, K., Li, J., Tang, T., Wang, X., Hou, Y., Min, Y., Zhang, B., Zhang, J., Dong, Z., Du, Y., Yang, C., Chen, Y., Chen, Z., Jiang, J., Ren, R., Li, Y., Tang, X., Liu, Z., Liu, P., Nie, J. & Wen, J. (2023). A Survey of Large Language Models. arXiv: 2303.18223. DOI: https://doi.org/10.48550/arXiv.2303.18223

Zhu, J. (2015). Optimization of power system operation. IEEE.

Zhu, J., Dong, H., Zheng, W., Li, S., Huang, Y. & Xi, L. (2022). Review and prospect of data-driven techniques for load forecasting in integrated energy systems. Applied Energy, 321, 119269.