

Electronic Supplementary Information

We provide here relevant dialogue between the user and ChatGPT. For full prompt history and code, we refer the reader to the following links:

- Python control class: <https://chat.openai.com/share/1a9afe99-86ac-499e-94a7-7d0a1a84aa23>
- GUI: <https://chat.openai.com/share/e02fff82-edea-45f3-964d-73ef2012f8ee>
- GUI (error handling): <https://chat.openai.com/share/057d1f7f-61a6-4ce0-9371-8f5b5267625c>
- Control class from the prompt template (Keithley 2400): <https://chatgpt.com/share/8d523f18-86c8-44af-8002-a5bcf45ffbb5>
- GUI from the prompt template: <https://chatgpt.com/share/b2abdd30-e66c-49db-af58-c3016f5837b0>
- Control class from the prompt template (sputtering instrument): <https://chatgpt.com/share/36e228c9-08bd-4dff-8e56-c73e988095fc>
- GUI from the prompt template (sputtering instrument): <https://chatgpt.com/share/aad1b352-6d9e-4c47-880b-b21f68505b0f>
- ChatGPT-crafted control code and GUI: https://github.com/NREL/Keithley_GPT
- Differential Evolution tool: <https://github.com/NREL/DE>

User: Great. I have a Keithley 2400 SMU, that can be controlled through SCPI commands. Using PyVisa, build a python class for controlling this instrument. We should be able to:

1. Choose whether to use the front or rear panel.
2. Use 2 or 4-wires measurement mode.
3. Perform an IV sweep, sourcing voltage and measuring current, with user-defined range for the sweep, and also user-defined measurement and sourcing range (each range has a different accuracy).
4. Define a current compliance of 10 mA (default), but also make this a variable.

Important to note that this script should be for a Keithley 2400. Browse the internet if needed to find specific commands for this SMU.

Do not try to run anything, just report the script back and I'll run it in the instrument and provide feedback, so we can adjust what's wrong. Thus, let's start from basic and initial configuration, such as connecting and sending commands to identify the connection address and the instrument, and then proceed through panel selection and so on. If something is not clear, ask for clarification.

ChatGPT: Certainly! To create a Python class for controlling the Keithley 2400 SMU using PyVISA and SCPI commands, we'll structure the class to include methods for the tasks you've described. I'll begin with a basic structure including initialization, connection, and identification methods, and then add methods for selecting the front or rear panel, setting measurement mode, performing an IV sweep, and defining current compliance. Before running this script, make sure you have PyVISA installed in your Python environment. You can install it using pip: `pip install pyvisa`

Also, you'll need the VISA backend, like NI-VISA or Keysight VISA, installed on your computer to interface with the instrument.

(Python code shown in Fig. 2 of the manuscript)

ChatGPT: This script provides a basic structure for your Keithley 2400 controller class. You'll need to replace 'GPIB0::24::INSTR' with the actual GPIB address or connection string of your Keithley 2400. Next steps would include adding methods for IV sweep with user-defined parameters. Let me know if this initial setup works for you, and then we can proceed with the implementation of the IV sweep functionality.

Figure S1: Dialogue between the user and ChatGPT outlining the development of a Python class for the Keithley 2400 SMU control via PyVISA and SCPI commands, including the ability to select panel types, measurement modes, perform IV sweeps, and set current compliance, followed by ChatGPT's affirmative response detailing the class structure and preliminary steps, emphasizing the necessity of PyVISA installation.

User: Great, it works. Now, implement methods to define current and voltage ranges, either for sourcing and measuring cases.

ChatGPT: Fantastic! Let's move on to implementing methods for defining current and voltage ranges in the Keithley2400Controller class. We'll add two sets of methods: one for setting the current range and another for setting the voltage range. Each set will include methods for both sourcing and measuring. The SCPI commands for setting the current and voltage ranges are typically structured as follows:

- For setting the source current range: `:SOUR:CURR:RANG <value>`
- For setting the source voltage range: `:SOUR:VOLT:RANG <value>`
- For setting the current measurement range: `:SENS:CURR:RANG <value>`
- For setting the voltage measurement range: `:SENS:VOLT:RANG <value>`

Here is how these methods can be implemented in the class:

```
class Keithley2400Controller:
    def __init__(self, resource_name='ASRL5::INSTR'):
        # ... existing initialization code ...

    def connect(self):
        # ... existing connection code ...

    def identify(self):
        # ... existing identification code ...

    def select_panel(self, panel='FRONT'):
        # ... existing code ...

    def set_measurement_mode(self, mode):
        # ... existing code ...

    def set_current_compliance(self, compliance):
        # ... existing code ...

    def set_nplc(self, nplc, measurement_type='CURR'):
        # ... existing code ...

    def set_source_current_range(self, range_value):
        self.instrument.write(f":SOUR:CURR:RANG {range_value}")

    def set_source_voltage_range(self, range_value):
        self.instrument.write(f":SOUR:VOLT:RANG {range_value}")

    def set_measure_current_range(self, range_value):
        self.instrument.write(f":SENS:CURR:RANG {range_value}")

    def set_measure_voltage_range(self, range_value):
        self.instrument.write(f":SENS:VOLT:RANG {range_value}")

# Example usage
controller = Keithley2400Controller()
controller.connect()
controller.set_source_current_range(0.1) # Set source current range to 0.1 A
controller.set_source_voltage_range(20) # Set source voltage range to 20 V
controller.set_measure_current_range(0.1) # Set measure current range to 0.1 A
controller.set_measure_voltage_range(20) # Set measure voltage range to 20 V
```

ChatGPT: Each of these methods takes a single parameter, `range_value`, which is the desired range for sourcing or measuring current or voltage. The units should be in amperes for current and volts for voltage. Test these methods with your Keithley 2400 SMU to ensure they work as expected. If you have any more functionalities to add or if any adjustments are needed, please let me know!

Figure S2: User-ChatGPT interaction to implement additional methods to set measure and source ranges for voltage and current, and resulting code.

User: Great, it works. Now, implement a method to perform an IV sweep, either by sourcing voltage or current, and either by measuring voltage or current. For example, this method should be capable to source voltage and measure current across a user-defined voltage interval, and the same for sourcing current and measuring voltage. Also, the step size of the sweep should be user-defined. For this task, it may be necessary to configure and initialize the instrument buffer with specific size. If you're not clear about this task, ask for clarification.

ChatGPT: I'm glad to hear that the previous implementations are working well. For the IV sweep functionality, we'll create a method that can handle both voltage sourcing with current measurement and current sourcing with voltage measurement. The method will allow users to define the start and end points of the sweep, the step size, and the type of sweep (voltage or current sourcing). In SCPI, setting up an IV sweep typically involves configuring the source function, setting the start and stop levels, the step size, and the measurement parameters. We also need to set up the buffer to store the measurement results. The buffer size will depend on the number of points in the sweep, which we can calculate from the start, stop, and step values.

Here's how you can implement the IV sweep method:

```
class Keithley2400Controller:
    # ... existing methods ...

    def iv_sweep(self, start, stop, step, source_type='VOLT', measure_type='CURR'):
        # Calculate number of points in the sweep
        num_points = int(abs(stop - start) / step) + 1

        # Set up source and measurement
        if source_type.upper() == 'VOLT' and measure_type.upper() == 'CURR':
            self.instrument.write(":SOUR:FUNC VOLT")
            self.instrument.write(f":SOUR:VOLT:START {start}")
            self.instrument.write(f":SOUR:VOLT:STOP {stop}")
            self.instrument.write(f":SOUR:VOLT:STEP {step}")
            self.instrument.write(":SOUR:SWE:RANG AUTO")
            self.instrument.write(":SOUR:SWE:SPAC LIN")
            self.instrument.write(f":SOUR:SWE:POIN {num_points}")
            self.instrument.write(":SOUR:SWE:DIR UP")
            self.instrument.write(":SENS:FUNC 'CURR:DC'")
        elif source_type.upper() == 'CURR' and measure_type.upper() == 'VOLT':
            self.instrument.write(":SOUR:FUNC CURR")
            self.instrument.write(f":SOUR:CURR:START {start}")
            self.instrument.write(f":SOUR:CURR:STOP {stop}")
            self.instrument.write(f":SOUR:CURR:STEP {step}")
            self.instrument.write(":SOUR:SWE:RANG AUTO")
            self.instrument.write(":SOUR:SWE:SPAC LIN")
            self.instrument.write(f":SOUR:SWE:POIN {num_points}")
            self.instrument.write(":SOUR:SWE:DIR UP")
            self.instrument.write(":SENS:FUNC 'VOLT:DC'")
        else:
            raise ValueError("Invalid source/measure type combination. Choose 'VOLT'/'CURR' or 'CURR'/'VOLT'.")

        # Initialize the buffer
        self.instrument.write(":TRAC:CLE") # Clear the buffer
        self.instrument.write(f":TRAC:POIN {num_points}")
        self.instrument.write(":TRAC:FEED:CONT NEXT")

        # Perform the sweep
        self.instrument.write(":INIT")
        self.instrument.write(":FORM:ELEM READ")
        return self.instrument.query_ascii_values(":TRAC:DATA?")

# Example usage
controller = Keithley2400Controller()
controller.connect()
sweep_data = controller.iv_sweep(start=0, stop=5, step=0.1, source_type='VOLT', measure_type='CURR')
print(sweep_data)
```

In this method:

- **start**, **stop**, and **step** define the sweep range and step size.
- **source_type** and **measure_type** determine whether you're sourcing voltage and measuring current, or sourcing current and measuring voltage.
- The buffer is set up to store the number of points calculated from the sweep parameters.
- The method returns the data from the sweep, which you can then process or plot as needed.

Figure S3: User-ChatGPT interaction to implement a current-voltage (IV) sweep method.

User: I'll give you the code for a backend control code named "Keithley2400Controller" stored in a file named "Keithley2400GPT.py". Your task is to build a professional-looking GUI for this code. Here are all things this GUI should accomplish:

- Connect to the instrument and check connection
- A frame for all setup parameters to be filled by the user, except overvoltage, which always defaults to 20 V in the backend code. These parameters are source type, measure type, start level, stop level, step level, compliance, nplc, source delay. For auto range, a toggle that defaults to auto range enabled should be present, but when it's disabled, the user should entry source and measure ranges.
- A button to perform the IV sweep based on all configuration parameters. The IV sweep should be executed asynchronously.
- plot area, to the right side of the configuration and setup frame. The data should be displayed only after the IV sweep is completed.
- Toggle to enabled y and x-axis log scale. Upon enabling/disabling this toggle, the plot should be dynamically updated.
- Button to save the data from the IV sweep. Voltage values in the first column, and current values in the second column.

Is it all clear? If so, I'll give you the code to control the instrument.

ChatGPT: Yes, I can help you design a GUI for your backend control code for the Keithley 2400. Please go ahead and share the code for the "Keithley2400Controller". This will help me understand how the backend is structured and how the GUI should interact with it. Once I have the code, I can provide you with a detailed design and even some sample code for the GUI using a Python library like Tkinter or PyQt, depending on your preference.

Figure S4: Initial prompt instructing ChatGPT to create a graphical user interface (GUI) for interfacing with the instrument's control class, to assess whether it correctly comprehended its task.

We briefly describe the features of our optimization tool, but more details can be found in the Github repository listed in the data availability section. We illustrate the usage of the optimization tool to find the minimum of the 2-dimensional domain Ackley function, which possesses many local minima and is widely used to benchmark optimization algorithms, where the goal is to find (x, y) that minimize the Ackley function defined in Eq. (1):

$$f(x, y) = -20 \exp \left[-0.2 \sqrt{0.5(x^2 + y^2)} \right] - \exp [0.5(\cos(2\pi x) + \cos(2\pi y))] + e + 20 \quad (1)$$

Fig. S5 a) shows a code snippet for the minimization of the 2D Ackley function. As long as the objective function is Numba-compatible and its arguments are `indv` only, any function can be passed to the `parallel` method from class `DE`. The search space is defined by setting lower and upper bounds for each element of the `indv` array, which contains the parameters of the objective function.

The term *individual* (`indv` in the code snippets) refers to the usual nomenclature in the differential evolution literature, but it is nothing more than a possible solution for the optimization problem — each component of an individual is a parameter that will be optimized. In the example of Fig. S5 a), `indv[0]` represents the x variable, while `indv[1]` is the y variable, so that we have the Ackley function, and the minimum and maximum allowed values of each dimension are defined by elements 0 and 1 of the lower and upper bound arrays. If the objective function needs more arguments or external data, such as voltage and current, we recommend using a function enclosure, where a higher-order function that can take additional arguments encapsulates the objective function, which serves as a closure, as shown in Fig. S5 b).

a) 2D Ackley Function Optimization

```
import DE
from numba import jit
import numpy as np

@jit(nopython=True)
def ackley(indv):
    term1 = -20 * np.exp(-0.2 * np.sqrt(0.5 * (indv[0]**2 + indv[1]**2)))
    term2 = -np.exp(0.5 * (np.cos(2 * np.pi * indv[0]) + np.cos(2 * np.pi * indv[1])))
    value = term1 + term2 + np.e + 20
    return value

lbound = np.array([-10,-10])
ubound = np.array([10,10])

score_list, solution_list = DE.parallel(ackley,
                                       runs=1, popsize=100, gmax=1000,
                                       lower_bound=lbound, upper_bound=ubound)
```

b) Objective Function with External Data

```
import DE
from numba import jit

def create_objective_function(voltage, current):
    """Create an objective function with external data."""
    @jit(nopython=True)
    def objective(indv):
        model = ...
        fitness_scalar = ...
        return fitness_scalar
    return objective

lbound = ...
ubound = ...

voltage, current = DE.load_raw_data("iv.txt", vmin=-2, vmax=2)
custom_objective = create_objective_function(voltage, current)

score_list, solution_list = DE.parallel(custom_objective,
                                       runs=1, popsize=100, gmax=1000,
                                       lower_bound=lbound, upper_bound=ubound)
```

Figure S5: a) Code snippet showing the usage of the optimization tool described in this work. In this example, the goal is to find the minimum of the 2-dimensional domain Ackley function, widely used to benchmark optimization algorithms. b) Definition of an objective function with external data (current and voltage) using a function closure, returning an objective function that depends only on the optimization individuals (`indv`), to keep the differential evolution nomenclature.

I have filled out a JSON file with details about my instrument and the required functionalities for Python-based control code development. Please use the information in the uploaded JSON file to:

1. Initial Setup: Based on the JSON details, recommend and provide the necessary Python libraries and a basic class structure to represent the instrument.
2. Communication and Basic Operations: Write code to perform the basic instrument functions and establish communication with the instrument using the specified communication interface and protocol. Implement the desired operations as outlined in the JSON.
3. Data Logging: Add functionality to log the data from the operations into the specified file format. Ensure the code handles reading from or writing to any external files mentioned in the JSON.
4. Error Handling and Logging: Incorporate robust error handling and logging mechanisms based on the details in the JSON. Ensure that errors are recorded and managed as specified.
5. Final Testing and Validation: Implement test and validation as outlined in the JSON file.

Please provide a final single script, making sure it is complete, functional, and ready for real-world use. Do not use placeholders, since a complete code with automated testing should be the output, and make sure all desired operations are implemented.

Important: cross-check the implemented methods with the desired operations in the JSON file, and add the functionalities that you eventually missed, presenting a fully corrected script.

Are all tasks clearly defined? If so, may I upload the JSON file?

Figure S6: Prompt template for developing code to control instruments using LLMs and a user-defined JSON file with desired capabilities.