Digital Discovery

ROYAL SOCIETY OF CHEMISTRY

PAPER

View Article Online
View Journal | View Issue

Check for updates

Cite this: *Digital Discovery*, 2025, **4**, 35

From text to test: AI-generated control software for materials science instruments†

Davi Fébba, Kingsley Egbo, William A. Callahan and Andriy Zakutayev

Large language models (LLMs) are one of the AI technologies that are transforming the landscape of chemistry and materials science. Recent examples of LLM-accelerated experimental research include virtual assistants for parsing synthesis recipes from the literature, or using the extracted knowledge to guide synthesis and characterization. However, these AI-driven materials advances are limited to a few laboratories with existing automated instruments and control software, whereas the rest of materials science research remains highly manual. AI-crafted control code for automating scientific instruments would democratize and further accelerate materials research advances, but reports of such AI applications remain scarce. The goal of this manuscript is to demonstrate how to swiftly establish a Python-based control module for a scientific measurement instrument solely through interactions with ChatGPT-4. Through a series of test and correction cycles, we achieved successful management of a common Keithley 2400 electrical source measure unit instrument with minimal human-corrected code, and discussed lessons learned from this development approach for scientific software. Additionally, a user-friendly graphical user interface (GUI) was created, effectively linking all instrument controls to interactive screen elements, and text prompts as well as JSON templates for interaction with ChatGPT are provided for this and other instruments. Finally, we integrated this AI-crafted instrument control software with a high-performance stochastic optimization algorithm to facilitate rapid and automated extraction of electronic device parameters related to semiconductor charge transport mechanisms from current–voltage (IV) measurement data. This integration resulted in a comprehensive open-source toolkit for semiconductor device characterization and analysis using IV curve measurements. We demonstrate the application of these tools by acquiring, analyzing and parameterizing IV data from a Pt/Cr_2O_3:Mg/β-Ga_2O_3 heterojunction diode, a novel stack for high-power and high-temperature electronic devices. This approach underscores the powerful synergy between LLMs and the development of instruments for scientific inquiry, showcasing a path to further accelerate research progress towards synthesis and characterization in materials science.

Received 29th May 2024
Accepted 22nd October 2024

DOI: 10.1039/d4dd00143e

rsc.li/digitaldiscovery

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM.
This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

1 Introduction

Recent advancements in artificial intelligence, particularly the emergence of Large Language Models (LLMs) such as OpenAI's GPT-3 and 4, have sparked the interest of the scientific community. A growing body of knowledge now evaluates LLMs for chemistry and materials science related tasks, such as chemistry knowledge,[1,2] text-parsing of solid state synthesis recipes,[3] domain-specific chatbots for scientific literature,[4] and also in fully autonomous experimentation workflows,[5] among many other applications.[6] ChatGPT, specifically, was recently included as a non-human addition to *Nature*'s 10 due to its profound impact on science in 2023.[7]

LLMs, endowed with natural language understanding and generation capabilities, have been particularly useful for developing computing code for a myriad of tasks,[8] and thus offer unprecedented opportunities for expediting the development of control solutions tailored to laboratory instruments, which traditionally involves intricate programming, extensive testing, and iterative refinement over prolonged periods. LLMs revolutionize this process by enabling researchers to articulate desired functionalities in plain language, thereby automating the generation of code snippets or complete control algorithms. Consequently, the development cycle is significantly compressed, allowing researchers to allocate more time to experimental design and scientific inquiry. Furthermore, the agility afforded by LLM-generated code empowers researchers to swiftly adapt control solutions to evolving experimental requirements or unforeseen challenges.

The integration of LLMs democratizes control solution development and fosters interdisciplinary collaboration in the

Materials Science Center, National Renewable Energy Laboratory (NREL), Golden, CO, 80401, USA. E-mail: Andriy.Zakutayev@nrel.gov; DaviMarcelo.Febba@nrel.gov

† Electronic supplementary information (ESI) available. See DOI: https://doi.org/10.1039/d4dd00143e

© 2025 The Author(s). Published by the Royal Society of Chemistry

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM.
This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

scientific community. These models provide a user-friendly interface that simplifies the conversion of ideas into functional code, enabling effective contributions from researchers with diverse backgrounds. Furthermore, applications developed using LLMs, particularly in Python, can amplify the application of the FAIR principles—data is Findable, Accessible, Interoperable, and Reusable. Python's comprehensive suite of open-source libraries inherently makes both the source code and output file formats open source, which significantly contributes to the FAIR data principles by ensuring that data handling practices are transparent and the generated data is readily accessible and useable by the wider research community.

The transformative benefits of LLMs in simplifying code development and enhancing data management seamlessly translate to specialized applications across scientific disciplines. Current–voltage (IV) characteristics, for example, play a pivotal role in understanding the fundamental properties and operational behavior of semiconductor devices, serving as a critical tool for diagnosing device performance, efficiency, and for identifying potential areas for optimization. For example, in solar cells, IV curves are essential for determining device-level parameters like open-circuit voltage, short-circuit current, and fill factor, which are key indicators of device efficiency,[9] and also for extracting parameters related to the fundamental physical processes, such as Shockley–Read–Hall recombination parameters.[10] Additionally, by modeling the IV curves with equivalent circuits and diode models, parameters such as reverse saturation current, diode ideality factor, series and shunt resistance, can be extracted.[11,12] Similarly, in diodes and transistors, IV curves help in assessing threshold voltage, leakage current, and on–off ratio, important parameters for evaluating device switching behavior and power consumption.[13–15]

IV measurement and analysis setups are widespread in materials science laboratories and are often automated using custom LabView or Python programs. Graduate students typically take on the task of automating these instruments, dedicating substantial time to navigating detailed reference manuals and developing and testing the control code. The analysis step of extracting physical parameters from IV curves also presents a significant challenge due to the nature of the equations resulting from equivalent circuit modeling. Traditional parameter extraction methods often involve tedious, manual adjustments and assumptions that can introduce errors or biases, making the process time-consuming and less accurate. Moreover, the sheer volume of data generated by modern semiconductor testing necessitates a more efficient approach to parameter extraction to keep pace with rapid technological advancements. Metaheuristic optimization algorithms, such as genetic algorithms, simulated annealing, and particle swarm optimization, provide a powerful solution to these challenges and have indeed been successfully applied to this problem in the field of photovoltaics.[16] By exploring the vast search spaces associated with IV curve analysis, these algorithms can find optimal or near-optimal parameter values that closely match experimental data, with no need to rely on any model assumptions, making these algorithms effective alternatives to automate the analysis of IV data.

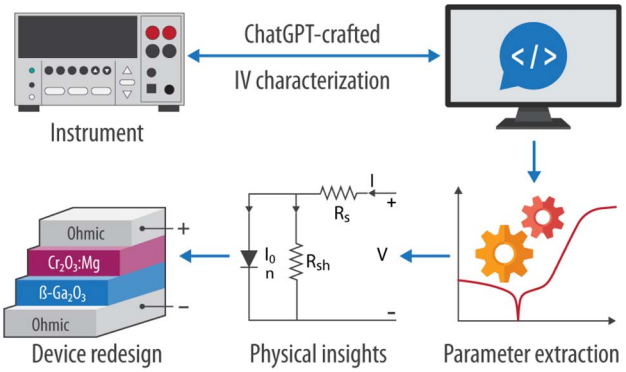

Fig. 1 Diagram depicting the AI-engineered current–voltage (IV) characterization module and parameter extraction tool described in this work. The Source Measure Unit (SMU) is connected to a control computer running a Python application developed by ChatGPT. The collected IV characterization data is subsequently processed using a high-performance differential evolution algorithm for automated parameter extraction from the IV curves, to provide insights for redesigning the device stack.

Here, we demonstrate the application of LLMs, specifically OpenAI's ChatGPT-4, to rapidly automate a Keithley 2400 Source Measure Unit (SMU), which is widely used for IV measurements, and create a graphical user interface (GUI) for it. The final control solution was accomplished without resorting to model fine-tuning, and serves as a compelling example of how LLMs can significantly lower the barriers to rapid automation in laboratory settings, especially for researchers with limited software development experience. Moreover, due to the versatile nature of LLMs, other characterization and synthesis instruments using standard commands for programmable instruments (SCPI), Modbus or other standard commands and protocols, can be automated by the same approach. We also address common pitfalls encountered during this process and present effective solutions, such as structured prompts and JSON file templates, with the objective of ensuring successful integration of LLMs for equipment automation.

We then integrated an evolutionary optimization algorithm into this measurement framework to extract semiconductor device parameters from IV measurements of electronic devices, such as those of diodes. This integration resulted in a comprehensive, end-to-end software solution that automates the entire process — from measuring current–voltage characteristics to extracting device parameters and gaining insights into semiconductor transport properties (Fig. 1). Finally, we applied this software solution to the characterization of Pt/Cr_2O_3:Mg/ß-Ga_2O_3 heterojunction diode operating under harsh environmental conditions. Given the widespread use of IV measurement methods and instruments discussed in this work, we have made the software available as open-source on GitHub, aiming to provide researchers with a solid foundation for more advanced electrical analysis.

© 2025 The Author(s). Published by the Royal Society of Chemistry

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM. This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

2 ChatGPT-crafted automation

2.1 Controlling the SMU

There are multiple approaches to code development, ranging from fully manual, human-driven methods to advanced AI-assisted systems. These include AI-aided dialogue, where users interact with the model to manually request and integrate code, as well as API calls to an LLM, which enable the AI to generate and directly insert code into projects. Other options include IDEs with AI integration that support in-environment code generation, and fully AI-managed platforms. Additionally, LLM-based software engineer agents can automate coding tasks,[17–19] offering further levels of automation.

We selected the AI-aided dialogue approach due to its simplicity, minimal setup, and the low level of coding expertise required. This method allows for rapid code generation and modification through natural language prompts, without the limitations on the number of LLM calls that can occur in some IDEs with LLM integration. Additionally, it gives the user the flexibility to choose any IDE, as the process is not tied to a specific development environment. To streamline our discussion, we focus on the overarching elements of code development and outcomes, with the comprehensive prompt history detailed in the ESI.†

Our initial inquiries evaluated ChatGPT's ability to generate Python code and its familiarity with the SCPI (Standard Commands for Programmable Instruments) protocol required for instrument control, as illustrated in Fig. 2. Despite including acronyms and an unintentional typographical error in our prompt, ChatGPT skillfully navigated these minor obstacles, accurately interpreting the context and providing relevant responses. Additionally, ChatGPT suggested using the PyVISA library, a Python interface to the Virtual Instrument Software Architecture (VISA) API, which is a standard framework for enabling communication between software applications and test and measurement equipment.

User: Can you run python code?

ChatGPT: Yes, I can run Python code. Feel free to ask for any kind of Python-related task, such as calculations, data analysis, generating graphs, or running simulations. Just let me know what you need, and I'll do my best to assist you!

User: Are you aware of the SCIPI for instruments, such as Keithley SMUs?

ChatGPT: Yes, I'm aware of SCPI (Standard Commands for Programmable Instruments), which is a standardized language used for controlling programmable test and measurement devices. SCPI is widely used in various instruments, including Keithley Source Measure Units (SMUs). Keithley SMUs, like many other test and measurement devices, often use SCPI commands for automation and control. These commands allow users to set up the instrument, perform measurements, and retrieve data in a standardized way, which is particularly useful in automated test environments. For example, in Python, you might use libraries like PyVISA to communicate with instruments using SCPI commands. PyVISA acts as an interface to the VISA library, which is an industry-standard API for communication with test and measurement equipment. With this setup, you can send SCPI commands to a Keithley SMU to configure it, initiate measurements, and read the results. If you have a specific task or question related to controlling Keithley SMUs or using SCPI commands in Python, feel free to ask!

Fig. 2 Initial prompts evaluating ChatGPT's comprehension of SCPI (Standard Commands for Programmable Instruments) and its proficiency in generating Python code. The dialogue demonstrates ChatGPT's resilience to a typographical error (SCIPI instead of SCPI), underscoring its robust understanding and ability to accurately respond to queries related to programming and instrument control.

Subsequently, we tasked ChatGPT with the development of a Python class for managing a Keithley 2400 SMU, detailed in Fig. S1 of the ESI.† We provided comprehensive instructions, outlined initial tasks, and suggested internet searches for additional information, leveraging the ability of ChatGPT-4 to augment its responses with web-based data. We also used standard terminology relevant to the field, including "4-wire measurement mode" and "compliance level".

Beyond a full comprehension of the task, ChatGPT proposed initial steps for creating a Python class to interface with the instrument, presenting a systematic approach for class development that includes initialization, connection, and identification, along with methods for instrument control. It also emphasized the critical requirement of installing PyVISA through `pip` and the need for a VISA backend, essential for enabling the Python scripts to communicate with the instrument. The complete ChatGPT's answer is shown in Fig. S1 of the ESI,† and the resulting Python code for this initial step in shown in Fig. 3. It is worth highlighting the understanding of SCPI commands and the requirement to encapsulate them in strings when transmitting them to the instrument through the PyVISA `write` method.

We tested this class and example usage, but changing the instrument's general purpose interface bus (GPIB) address to the correct one in the `resource_name` argument, and we successfully were able to select the front panel, change the measurement mode to a 4-wire mode, and set a current compliance of 10 mA. Besides the instrument's address—which we did not pass to ChatGPT and depends on port addressing—no correction had to be made on the code, and no errors were found during execution, which is notable for a first trial. After this initial test, we informed ChatGPT about the instrument's GPIB address and asked for methods to set current and voltage source and measure ranges, which were successfully implemented and tested, again with no errors. This step is shown in Fig. S2 of the ESI.†

The next step was to ask ChatGPT to implement the most challenging part of the control class: a current–voltage (IV) sweep. Given the complexity of this task, we provided specific requirements, such as what it should source and measure, buffer set up and initialization, and so on, as shown in Fig. S3 of the ESI.† The resulting code missed the command to enable output on the instrument and the previously created methods were not used. After a few iterations to correct these issues and an unexpected output upon testing the code, we finally decided to provide ChatGPT with an example from the instrument's manual on how to program a linear IV sweep. Details about these interactions can be found in the prompt history.

The IV method was then refactored to take the provided example into account, but accounted only for sourcing current

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM.
This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

```python
import pyvisa

class Keithley2400Controller:
    def __init__(self, resource_name):
        self.resource_name = resource_name
        self.instrument = None
        self.current_compliance = 0.01  # 10 mA default

    def connect(self):
        rm = pyvisa.ResourceManager()
        self.instrument = rm.open_resource(self.resource_name)
        self.instrument.write("*RST")  # Reset the instrument
        self.instrument.write("*CLS")  # Clear the status

    def identify(self):
        return self.instrument.query("*IDN?")

    def select_panel(self, panel='FRONT'):
        if panel.upper() == 'FRONT':
            self.instrument.write(":ROUT:TERM FRON")
        elif panel.upper() == 'REAR':
            self.instrument.write(":ROUT:TERM REAR")
        else:
            raise ValueError("Invalid panel option. Choose 'FRONT'
            ↪ or 'REAR'.")

    def set_measurement_mode(self, mode):
        if mode == 2:
            self.instrument.write(":SYST:RSEN OFF")  # 2-wire mode
        elif mode == 4:
            self.instrument.write(":SYST:RSEN ON")  # 4-wire mode
        else:
            raise ValueError("Invalid measurement mode. Choose 2 or
            ↪ 4.")

    # Placeholder for IV sweep method
    # Placeholder for setting sourcing and measurement ranges

    def set_current_compliance(self, compliance):
        self.current_compliance = compliance
        self.instrument.write(f":SENS:CURR:PROT {compliance}")

# Example usage
controller = Keithley2400Controller('GPIB0::24::INSTR')  # Replace
↪ with your actual GPIB address
controller.connect()
print(controller.identify())
controller.select_panel('FRONT')
controller.set_measurement_mode(4)
controller.set_current_compliance(0.01)  # 10 mA
```

Fig. 3 ChatGPT-crafted Python class (initial architecture, before subsequent refactoring based on prompts) for interacting with a Keithley 2400 SMU, with methods for instrument connection, panel and measurement mode selection, and setting a compliance level. An example usage was also provided.

and measuring voltage, and left no room for range options, which was in fact expected, since ChatGPT only reproduced the example. We then instructed ChatGPT to modify this method so we could source and measure both current and voltage, and to not forget to leave compliance, sourcing and measuring ranges, and number of power line cycles (NPLC) as variables, besides making sure to use the methods it had already developed.

After another round of code refactoring, we had a functional version of the IV sweep method. During testing, the instrument initiated measurements, but PyVISA reported a timeout error. We reported this issue to ChatGPT, which then updated the controller class by adding a timeout parameter, set to a default of 25 seconds, effectively resolving the problem. Following a few additional instructions to manage the resulting data—with ChatGPT executing code online to understand and format the data—and incorporating parameters such as compliance, overvoltage protection, and range levels, ChatGPT finally produced a fully operational Python class to interact with the Keithley 2400 SMU.

We needed to implement only two code modifications. The first addressed auto-ranging during the IV sweep. We noticed the sweep command necessitated a specific instruction for selecting between a fixed range, as pre-configured in the instrument, and auto-range. This aspect was not initially managed correctly by ChatGPT due to lack of proper instructions, so a minor and manual correction was made. The second modification involved the calculation of sweep points. Specifically, ChatGPT overlooked the necessity of using the absolute value for both the difference between the start and stop levels and the step size, which is crucial for ensuring the linear sweep operates correctly in both ascending and descending modes. Although the inclusion of `abs(stop — start)` was correct in early iterations of the sweep method, it was inadvertently omitted following subsequent code refactorization. Detailed information about the prompts and the generated Python scripts are accessible *via* the links in the ESI.†

Finally, although we did not specify anything regarding error handling, it is interesting to note ChatGPT's behavior when dealing with possible errors. For instance, specifying anything different from 2 or 4 for the measurement mode will raise an error since these are the only two allowed parameters, and ChatGPT properly handled this issue, as shown in Fig. 3. On the other hand, the `connect`, `identify`, and `iv_sweep` methods did not handle errors, and also the methods defining range parameters — which was expected since we did not provide any information regarding allowed values from the instrument manual. The final control code was subsequently applied to the characterization of a heterojunction device, as detailed in Section 5, and no issues were encountered after connecting the instrument to the control computer. This iterative approach to prompt-based software development with ChatGPT highlights the need for user review of the generated code, and acts as a safeguard to ensure proper functionality, while still offering significant time improvements over manual code development.

2.2 Graphical user interface (GUI)

GUIs are especially useful for quick measurements and for most of materials science researchers that are usually not familiar with scripting programming languages. Thus, we decided to instruct ChatGPT to design and implement a minimal graphical user interface for the instrument control class. For that, we started a new chat and directed ChatGPT to develop a GUI for our instrument control module. We focused on outlining the required functionalities and ensuring ChatGPT comprehended the project scope, as illustrated in Fig. S4 of the ESI.†

We copied the control module to the prompt, after which ChatGPT answered with a design for the required application and started generating a code snippet, but mentioned that "It doesn't include detailed implementations of event handlers and plotting", as recorded in the prompt log. Given that, we stopped the answer and asked ChatGPT to provide a detailed and complete implementation. After this additional instruction, ChatGPT answered with a step-by-step breakdown of the code, as shown in the prompt history available in ESI.†

© 2025 The Author(s). Published by the Royal Society of Chemistry

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM.
This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

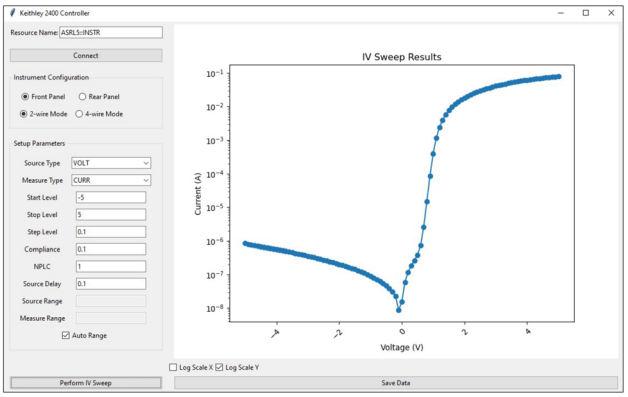

Fig. 4 ChatGPT-crafted graphical user interface for a Keithley 2400 controller. The user can select the operation panel, measurement mode, source and measure variables (voltage and current), sweep parameters (start, stop, step size), compliance, source and measurement ranges, besides NPLC and delay parameters. The data resulting from the IV sweep is plotted and can be saved through a file dialog.

Our previous attempts to develop this GUI were functional but visually unappealing, featuring misplaced buttons and excessive empty spaces. Learning from these experiences, we provided detailed specifications for layout, frame alignment, and button placement, which led to an extensive conversation until the GUI was functional and aesthetically pleasing, as evidenced by the chat log in the ESI materials.†

Finally, ChatGPT did not handle error exceptions well for the GUI, since the IV sweep and save data buttons were still enabled even if the instrument was not connected and no data had been collected. Thus, we provided the full GUI code to ChatGPT and asked for appropriate error handling measures for the connect button, save data and perform IV sweep buttons, specifying that these buttons should be disabled if the instrument is not connected, and that the save data button should be disabled if no data were collected. We also required the same for panel and measurement mode toggles. All of these additional features were successfully implemented, as recorded in the prompt history available in the ESI,† and we updated the GUI code. The final GUI, shown in Fig. 4, features all necessary control parameters for acquiring current and voltage data from the instrument using linear sweeps, including panel selection, measurement mode (2 or 4-wires), range specifications (including auto-range), compliance, measurement speed and source delay. Moreover, the resulting data is displayed on the screen, and the user can save data as a text file.

3 Discussion

3.1 Protocol for code development

Aided by ChatGPT, we created a prompt template, as shown in Fig. S6 of the ESI,† and a customizable JSON file to streamline the development of control code for laboratory instruments using LLMs. The JSON format, with its key–value pair structure, was chosen for its clarity and ease of use, making it accessible to non-experts and easily interpretable by ChatGPT. This framework encompasses most aspects of code generation, from setup to validation, including error handling. It also facilitates the creation of a Python class for instrument control by providing all necessary details on how the code should interface with the instrument. This includes specifying relevant Python libraries and dependencies, communication interfaces (*e.g.*, GPIB, TCP/IP), protocols (*e.g.*, SCPI, Modbus), external configuration files, and desired functionalities, all of which ChatGPT uses to generate the Python code. Additionally, we developed a prompt template for GUI generation, as shown in Fig. 5, to guide ChatGPT in mapping the methods and variables of the

I have a Python script that includes a class responsible for controlling an instrument. This class has several methods and variables that need to be mapped to a GUI. Please develop a fully functional GUI based on the following criteria:

1. **Class Methods and Variables**:
   - Each method in the class should be represented by a button or toggle in the GUI.
   - All class methods and variables should be directly mapped to appropriate input fields (e.g., text boxes, sliders, dropdown menus) in the GUI, allowing users to modify their values.
   - The GUI must include error-handling for all buttons and toggles to manage invalid inputs or failed operations.
   - Methods that return values should have corresponding output display elements (e.g., labels or text areas).
   - Boolean variables should be represented as checkboxes or toggle switches.
   - Variables with limited options should be represented as dropdown menus.
2. **GUI Layout and Design**:
   - Please determine the most effective way to map these methods and variables to screen elements, ensuring the layout is user-friendly and logical.
   - The GUI should feature a clean and intuitive design, with sections clearly labeled for different functions and settings.
3. **Error Handling**:
   - Implement validation for all input fields to ensure that users enter valid data.
   - The GUI should provide clear and informative error messages when inputs are invalid or when an operation fails.
4. **Additional Features**:
   - Add tooltips to buttons and input fields to provide users with additional information about their functions.
   - If applicable, include a real-time display of instrument status or data.
5. **Final Script Requirements**:
   - Ensure that the generated GUI script is complete and functional, with no placeholders or incomplete sections.
   - The GUI script should directly import the instrument class from the provided Python file and use its methods to ensure that the GUI operates as intended.

Once the script is uploaded, please analyze it and generate the corresponding GUI code in Python using an appropriate framework (such as Tkinter, PyQt, or another suitable library). Ensure that every aspect of the GUI is fully implemented so that the resulting script is ready to run without requiring any further modifications.

Fig. 5 Template for developing a GUI based on a Python class: this template outlines the key criteria and guidelines for creating a fully functional GUI that maps methods and variables from a Python class to appropriate screen elements. The template includes considerations for error handling, layout design, and additional features to ensure that the resulting application is operational.

© 2025 The Author(s). Published by the Royal Society of Chemistry

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM.
This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

instrument control code to GUI elements such as buttons, toggles, and text areas.

We validated this framework by developing basic methods and functional GUIs for the same SMU and for a thin film deposition instrument (details about this instrument be found in a previous work[20]), each using distinct communication interfaces and protocols: GPIB and SCPI for the SMU, and TCP/IP with open platform communications unified architecture (OPC UA) for the deposition instrument, besides an external configuration file. For the purpose of validating the suitability of this framework and ChatGPT capabilities to handle external files, we limited this validation to basic instrument actions and GUI elements. Functional applications were rapidly developed, meeting most of the design specifications, including automated testing, data logging, and error handling. In both cases, ChatGPT accurately analyzed the JSON files and prompted the user to upload additional configuration files when needed. Expanded functionalities can also be developed with this framework, but we anticipate that further prompt interactions with ChatGPT will be necessary to refine and review more complex features. The prompt and JSON templates, along with the resulting control classes and GUIs, are available in ESI.†

3.2 Lessons learned and pitfalls

In our example of automating a Keithley 2400 SMU, we instructed ChatGPT to develop all control methods within a class-based structure, allowing for easy integration of new methods and enabling the instrument class to be used in more complex, multi-instrument applications. We recommend starting with basic functionalities and gradually expanding to more advanced features, ensuring a solid foundation, minimizing errors, and making the code more maintainable and scalable as the project grows.

ChatGPT demonstrated a good overall understanding of the required tasks to automate the SMU. Simple methods, such as panel selection and measurement mode, were implemented effectively. However, despite our initial instructions specifying that the control code should perform IV sweeps and set instrument ranges, ChatGPT initially overlooked these requirements and left placeholders for these methods, as shown in Fig. 3. We also observed redundancy in the code due to oblivion of functions that were already defined. Consequently, we had to explicitly request the inclusion of these omitted functionalities and ensure the use of previously developed functions. For the IV sweep method, once we provided an example from the instrument manual, ChatGPT successfully developed a function to replicate the instructions from the example and mapped variable values to function arguments. This highlighted that learning through examples is an effective way for LLMs to quickly achieve desired functionalities.

The GUI development was particularly challenging. Although ChatGPT correctly mapped the instrument parameters to screen elements, it was constantly using placeholders for methods that should be defined, and omitting parts of the code that it considered to not have changed between prompt iterations, which can be confusing when only specific sections are corrected. Furthermore, the oblivion problem resulted in objects being used without having been defined. The adjustment of screen elements to match a user-defined layout was also possible, but we found that fine adjustments, such as updating the plot ticks to scientific notation to prevent overlapping and dynamic update of scales were difficult to implement with few prompts.

Error-handling in the generated code was inconsistent and insufficient. For instance, methods that interface with the instrument require an active connection to function properly, yet ChatGPT failed to implement connection checks. Therefore, we recommend emphasizing the importance of robust error-handling and additional validation checks when using ChatGPT for equipment automation. Properly addressing various potential issues is crucial, not only to prevent damage to expensive instruments but also to improve code readability and facilitate debugging. Although our control code did not initially include validation mechanisms and automated testing, we strongly advise incorporating comprehensive testing, which can also be facilitated by ChatGPT.

We also noticed that if the prompt history becomes long or if the user focus on specific sections of the code for further corrections, ChatGPT-4 may forget about previous methods and variables, which can lead to execution errors or redundancy. Moreover, several code revisions by ChatGPT may inadvertently lead to the loss of some essential methods and functionalities, which requires detailed review.

Using the prompt template and JSON file significantly streamlined code development compared to specifying details on-the-fly in a purely dialogue-based approach. The user can provide all specifications upfront, guiding ChatGPT with clear constraints and desired capabilities in an organized manner that is easily comprehensible. While the template aided in GUI development, some critical control code functionalities were initially overlooked. By prompting ChatGPT to reassess the code and identify these gaps, it engaged in a self-reflection process[21] essential to refining the final application. This iterative approach—reviewing the JSON-defined features, comparing them with the code, and resolving issues through conversational feedback—proved crucial for successful development.

However, the efficacy of this approach heavily depends on the details provided in the JSON file and the structure of the instrument control class. For instance, creating a GUI from an unstructured Python file with no classes, methods, or error handling mechanisms makes it challenging to map instrument functionalities to screen elements. During control class development, ChatGPT will implement only the methods specified in the JSON file, making it imperative to clearly define methods and functionalities to achieve accurate results.

Although it is certainly possible for researchers with no previous code experience to learn from ChatGPT how to start developing an application from scratch, including installing a proper integrated development environment (IDE) and language packages, we found that code review by a researcher with programming background was an important part of the process. Although ChatGPT developed all the code, human review detected flaws that could otherwise go unnoticed. In the

© 2025 The Author(s). Published by the Royal Society of Chemistry

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM.
This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

dialogue-based approach, ChatGPT-crafted code must be copied to another environment for code execution. Thus, we recommend using IDEs with extensions with code assistance features, such as highlighting undefined variables, improperly imported libraries and syntax issues, or using IDEs with LLM integration, such as *Cursor*,[22] to mitigate these issues.

To ensure a more robust code review, this step could be coupled to a version control tool, where an expert would review the code before deployment. For maintainability and scalability, particularly as the complexity of laboratory equipment and experiments increases, leveraging IDEs with LLM integration or tools like *LangChain*[23] for automating code generation, testing, and human-aided review would be beneficial. Additionally, semi-autonomous software engineer agents, such as *Devin*[24] and *Amazon Q*,[25] can further automate many parts of the code development lifecycle. However, for small laboratory applications such as automation of single instruments with straightforward processes, the interactive dialogue-based approach—with prompt templates and JSON specification file—is a fast solution if human-review is readily available.

In this work, both the Keithley SMU and the thin film deposition instrument, for which ChatGPT developed code, rely on well-documented communication protocols that are supported by Python libraries, making it feasible to create effective applications. This is because the relevant data is likely included in the LLM's training corpus. However, the use of proprietary protocols—particularly those without Python library support or limited public documentation—can lead to code hallucinations. In these scenarios, ChatGPT might attempt to call nonexistent methods or import unavailable libraries. Moreover, graphical programming languages like LabView present challenges for ChatGPT-driven automation; it cannot execute LabView code or accurately render data flow and block diagrams, complicating the process of reporting results back to ChatGPT for debugging.

The field of LLM-assisted software development is evolving rapidly, with significant advancements occurring even during the brief review period of our work. Recent publications from August and September 2024 (ref. 17–19) and the release of OpenAI's o1 models in September have enhanced capabilities in this domain. Innovations such as LLMs optimized for code generation, extended context windows to prevent omissions, and automated iterative reasoning—introduced in the new ChatGPT o1 models—are improving the accuracy, completeness, and efficiency of the code generation process.

4 IV Parameter extraction

4.1 Streamlining the parameter extraction process through an open-source tool

To automate and speed up the parameter extraction from IV curves, we developed an open-source optimization tool that leverages the self-adaptive differential evolution (DE) algorithm[26]—benchmarked on the photovoltaics (PV) parameter extraction problem and proven robust against noise[27]—and Numba, a high-performance Python compiler that allows easy code parallelization across multiple CPUs.[28]

The DE parameter extraction tool, which was not developed by ChatGPT, can be applied to single-objective optimization problems by simply defining an objective function and a search space that are then passed to the algorithm. Although other techniques, such as Newton–Raphson, gradient-based techniques, or random sampling could be applied to this problem, they either rely on initial points or derivatives, or do not balance exploration/exploitation as well as the DE algorithm implemented here.[12,29,30] Despite many implementations of the standard differential evolution algorithm are available, such as in *scipy* and *Mathematica*, we have not found any open-source Python implementation of the self-adaptive DE algorithm—only a `C++` implementation (pagmo[31]) was found. In principle, it should be possible to translate the `C++` implementation of this code into a Python implementation using LMMs. While an interesting application, this translation was not attempted in this work and will be explored in future studies. Extensive discussions and an application of self-adaptive DE algorithm to the PV parameter extraction problem, including convergence curves, can be found elsewhere.[26,27,32]

Our optimization tool leverages the easy parallelization enabled by Numba: the parameter `runs` controls the number of parallel optimization processes, each starting with a different `seed` parameter, so that each *evolution* of parameters starts at different points, to enhance the likelihood of finding the global optimum. We measured the optimization performance of our tool on the 2-dimensional domain Ackley function, which is discussed in the ESI† as an application to a general optimization problem, on a laptop equipped with a 11th Gen Intel Core i7-11850H (2.50 GHz) and 16 GB of RAM, running a 64-bit Windows 10 OS: it took on average 3.9 seconds to run 10 parallel optimization routines (across 100 code executions), each with a population size of 100 individuals evolving until a maximum number of 10 000 generations is reached. The results available in the Git repository (`ackley_results.py`) demonstrate that the algorithm successfully found the minimum of the 2D Ackley function, which is 0 at $(x, y) = (0, 0)$, by consistently finding x values between -3.6×10^{-16} and 3.6×10^{-16}, and y values between -3.6×10^{-16} and 3.6×10^{-16}, with the minimum of the Ackley function always converging to -4.4×10^{-16}, indicating that the errors were negligible.

4.2 Parameter extraction methods

The transport properties in the fabricated devices can be studied through diode parameters extracted from IV measurements, usually through the single-diode model, defined by eqn (1) with a shunt term:

$$I = I_0\left[\exp\left(\frac{V - IR_s}{nV_T}\right) - 1\right] + \frac{V - IR_s}{R_{sh}}, \quad (1)$$

where I_0, n, R_s and R_{sh} represent the diode reverse saturation current, ideality factor, series and shunt resistance, respectively, and $V_T = k_\beta T/q$ is the thermal voltage. Hence, from each set of IV measurements at a given temperature, the goal is to extract the four diode parameters from eqn (1). The optimization tool described in the previous sections can be applied to this problem by converting it into an optimization problem, where

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM. This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

the goal is now to minimize an error metric between experimental and simulated IV curves, which depend on the diode parameters: $I = f(V, V_T, I_0, n, R_s, R_{sh})$. Thus, the set of diode parameters providing the lowest error metric is considered to better describe the experimental data.

To simulate IV curves and extract diode parameters, we first rearrange eqn (1) as a function of voltage and single-diode parameters only, through the Lambert W function,[33–35] as defined by eqn (2):

$$I = \frac{nV_T}{R_s} W_0\left\{\frac{R_s I_0}{nV_T(1+R_s/R_{sh})}\left[\exp\frac{R_s I_0 + V}{nV_T(1+R_s/R_{sh})}\right]\right\} - \frac{I_0 - V/R_{sh}}{1+R_s/R_{sh}}, \quad (2)$$

where W_0 is the main branch of the Lambert W function. Additionally, we considered a voltage offset $V = V + V_{offset}$ to account for additional voltage inaccuracies from the measurement setup. To compare how close simulated curves are to the experimental ones, we define an objective function, which will be minimized by the optimization algorithm. In this case, we used the root mean squared error, but taking the relative deviation to account for the orders of magnitude variation in current, as defined by eqn (3):

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{I_{i,exp} - I_{i,sim}}{I_{i,exp}}\right)^2}, \quad (3)$$

where $I_{i,sim}$ is the i-th simulated current point obtained from eqn (2), $I_{i,exp}$ are the experimentally measured current points and N is the number of experimental IV pairs.

Therefore, the goal of the optimization algorithm is to find single-diode parameters that minimize the RMSE. However, since we used Numba to compile the DE algorithm, we can only use Numba-compatible functions when defining an objective function. Since off-the-shelf implementations such as `scipy.special.lambertw` are not Numba-compatible, we approximated the main branch of the Lambert W function, $w = W_0(x)$, through the Newton–Raphson method[36] as given by eqn (4):

$$w_{i+1}(x) = w_i(x) - \frac{w_i(x)e^{w_i(x)} - x}{e^{w_i(x)} + w_i(x)e^{w_i(x)}}, \quad (4)$$

where a rough initial guess is given by $w_i(x) = \ln(x+1)$. In this case, we found that 10 iterations were sufficient to approximate W_0 within an absolute error lower than 10^{-10} in the interval $[10^{-10}, 10^6]$. An implementation to extract single-diode parameters from IV data using the applications reported here is provided in our remote repository (ESI†), as well as the measured IV data and the results of the Newton–Raphson approximation of the Lambert W function.

5 Application: modeling Pt/Cr_2O_3:Mg/β-Ga_2O_3 heterojunction diodes

We present here an application of the developed control code and streamlined optimization tool: the modeling of Pt/Cr_2O_3:Mg/β-Ga_2O_3 heterojunction devices. Since the most important parameters regarding the electrical performance of these devices are obtained from IV curves, and diode parameters can be extracted from the experimental IV curves using metaheuristic algorithms, both tools developed in this work are readily applicable.

5.1 Device fabrication and IV measurements

A Pt/Cr_2O_3:Mg/β-Ga_2O_3 vertical heterojunction diode was fabricated using an 7 μm lightly Si-doped (8×10^{-15} cm^{-3}) n-type β-Ga_2O_3 drift layer grown on a conductive bulk (001) β-Ga_2O_3 substrate (NCT). The schematic of the device cross-section is shown in inset of Fig. 6. Before device fabrication, photoresist was removed from the as-delivered substrates *via* an organic wash followed by a piranha rinse. Large area ohmic contact of Ti/Au (5 nm/100 nm)[37] was deposited on the backside of the substrate *via* e-beam evaporation, followed by a rapid thermal annealing (RTA) in N_2 ambient at 550 °C for 1 min. Next, the Cr_2O_3:Mg layer was grown by pulsed laser deposition (PLD), a ceramic target of 8 at% Mg-doped Cr_2O_3 was ablated using a pulsed KrF excimer laser at a frequency of 10 Hz and energy of 300 mJ. Growth was performed at a substrate temperature setpoint of 600 °C and O_2 partial pressure of 3×10^{-4} torr. Details of the electrical properties of the Cr_2O_3:Mg layer have been described elsewhere.[38] To make the top contact shadow masks were used to define an array of 1 mm diameter pads and e-beam evaporation was used to deposit a 30 nm Pt layer on the Cr_2O_3:Mg.

Current–voltage data from a Pt/Cr_2O_3:Mg/β-Ga_2O_3 heterojunction diode were obtained in a high temperature and highly-automated probe station capable of controlling a reducing atmosphere for temperature-dependent performance and time-

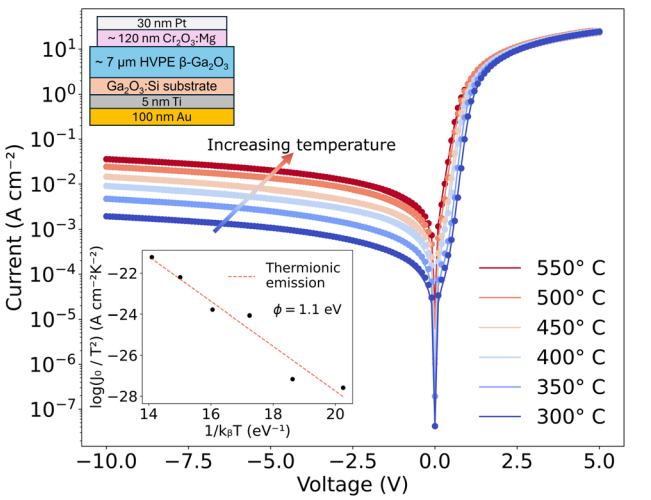

Fig. 6 Architecture of a Pt/Cr_2O_3:Mg/β-Ga_2O_3-based heterojunction diode and corresponding current–voltage data measured at several temperatures. Points represent experimental data and the solid lines represent the fits with the best set of single diode parameters that models the data. MAPEs for ranged between 4.7% and 7.8%. The inset shows that J_0 can be modelled by the thermionic emission model, yielding a barrier height of 1.1 eV.

© 2025 The Author(s). Published by the Royal Society of Chemistry

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM.
This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

dependent reliability electrical device measurements. All IV data were obtained with the ChatGPT-developed control module for the Keithley 2400 SMU. In this application, we sourced voltage between −5 V to 10 V and measured the resulting current in a 2-wire measurement mode, under normal measurement speed (NPLC = 1). Futhermore, we set current compliance to 200 mA and used auto-range for both source and measure functions. A single heterojunction diode was measured at various temperatures without H_2 reducing conditions, and at 500 °C under different gas flow conditions. Two Alicat mass flow controllers (MFCs) were used to control the flow of N_2 and forming gas (2% H_2), which were directed into the test chamber from two separate gas lines that mixed before the chamber feedthrough. The total gas flow was always 100 sccm, with the forming gas flow set at 0 sccm, 2 sccm, and 5 sccm. JV curves were recorded at 1 min intervals without any initial device equilibration time and were subsequently analyzed using a parameter extraction procedure, as detailed in the following section.

5.2 Results

The developed tool described in this work makes it straightforward to extract diode parameters from IV curves by just defining search bounds and eqn (3) as a custom objective function through a function closure, as shown in Fig. S5b†, and in the parameter extraction script available in our Git repository. Thus, IV data obtained with the ChatGPT-crafted control module is piped to the parameter extraction tool, and in a few seconds all relevant diode parameters are available.

For this application case, Fig. 6 shows the IV data measured at several temperatures and the resulting fitting from −10 V to 5 V for a Pt/Cr_2O_3:Mg/β-Ga_2O_3-based heterojunction diode in the 300 °C −550 °C temperature range under trace H_2 concentration, using the ChatGPT-crafted control module for Keithley 2400 SMUs. Mean absolute percentage errors (MAPE) ranged between 4.7% and 7.8%, demonstrating that the parameter extraction tool was able to successfully extract diode parameters from the IV curves.

As shown in the inset of Fig. 6, the extracted reverse saturation current J_0 can be modelled by the thermionic emission model, defined by eqn (5):

$$J_0 = A^* T^2 e^{-\phi/k_B T}, \quad (5)$$

where A^* is the effective Richardson constant and ϕ is the barrier height, found to be 1.1 eV for this device, which is similar to what has been reported for β-Ga_2O_3 Schottky devices[39,40] and lower than NiO/β-Ga_2O_3 heterojunction devices.[41] Poole–Frenkel emission was found to account for the electric field dependence of J_0 in Pt/Cr_2O_3:Mg/β-Ga_2O_3 devices, with unintentional Mg-rich layer at the heterojunction interface.[38] Since both thermionic emission and trap-assisted transport can explain the temperature- and field-dependence of current, it appears that both transport mechanisms are present, *i.e.*, activation from the conduction band over the barrier at the junction, and activated transport through trap states in the Mg-rich interfacial layer.

Moreover, we investigated the device's response to H_2 atmosphere conditions, by extracting the diode parameters at several H_2 concentrations over time, as shown in Fig. 7. Fig. 7a shows the reverse saturation current extracted from IV curves measured at different H_2 concentrations as a function of time. The increase in J_0 with H_2 can be attributed to a decrease of the potential barrier for thermionic emission.[42] On the other hand, the steady rise in J_0 over time stems from the device's inability to attain an equilibrium state within the observed time frame for the applied hydrogen concentrations, likely due to the large volume of the measurement chamber. Moreover, a departure from ideal diode behavior as H_2 concentration increases (Fig. 7b), with a corresponding increase in lumped series resistance (Fig. 7c), and a decrease in shunt resistance (Fig. 7d) show that, due to the catalytic nature of the Pt contact, the device responds to exposure to H_2, resulting in a corresponding change in electrical parameters.

We measured the performance of our software package for this parameter extraction problem with the same laptop used to minimize the Ackley function, and it took on average 58.7 seconds to run 10 parallel optimization routines with external data reading (single IV curve), each with a population size of 100 individuals evolving until a maximum number of 10 000 generations is reached. We note that the computing time will change according to the calculations required to compute the objective function. Here, eqn (4) is used to approximate the Lambert *W* function for every voltage point for each individual, which considerably increases the computational burden — as a comparison, it took less than 10 seconds to minimize the Ackley function for the same optimization parameters. Still, 10 parallel evolutionary optimization processes can be completed in under 1 min.

Finally, the amount of data can quickly increase for reliability analysis, when IV curves are measured over extended periods of time under several gas concentrations. For these

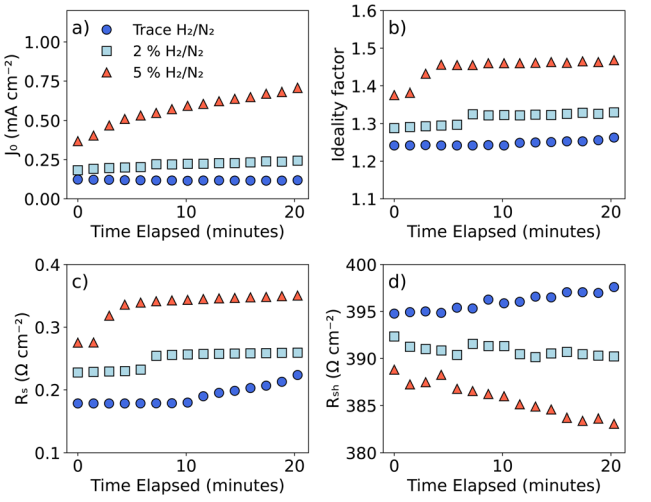

Fig. 7 Single-diode parameters as a function of time and forming gas percentage in the N_2 + H_2/N_2 mixture for the tested Cr_2O_3/β-Ga_2O_3-based heterojunction diode. Device degradation was observed upon exposure to H_2/N_2 while enduring a temperature soak at 500 °C.

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM.
This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

cases, a manual parameter extraction based on curve fitting becomes impractical. Hence, having a tool that can quickly extract diode parameters with minimal user effort can considerably help speed up the characterization process, and take the human out of the loop for fully autonomous accelerated degradation studies of electronic devices. Finally, we mention that complete implementation examples and the code used to generate the figures shown in this work can be found in the ESI.†

6 Conclusions

In this study, we demonstrated how to develop control software for materials science instruments using LMMs. As an example, we automated a Keithley 2400 Source Measure Unit (SMU) through an interactive dialogue with ChatGPT-4, that required just a few hours of time and no human-developed code besides few minor corrections. This software development method significantly streamlined the instrumental setup and testing phases, allowing researchers to focus on getting and analyzing materials science and device engineering results, rather than grappling with programming tasks. Furthermore, the development of a graphical user interface (GUI) as part of this automation process, which is notorious for having a tedious software development process, enhanced the user experience with the measurement instrument, making it more intuitive and accessible than the programming language, which is especially beneficial for materials science researchers with little scripting practice. To facilitate the automation of laboratory instruments, while preventing the shortcomings observed during on-the-fly requirements in a purely conversational approach, we developed prompt templates and a JSON file to provide detailed specifications to guide ChatGPT in the development process, and observed that it resulted in accurate code matching user-defined features.

Furthermore, we developed a Python-based implementation of the self-adaptive differential evolution algorithm for parameter extraction analysis of IV electrical measurement results, considering the widespread use of this data analysis approach. This analysis software implementation is enhanced by Numba, a just-in-time compiler that transforms Python code into machine code, significantly accelerating the parameter extraction process from IV curves. By showing a real-world application of this software platform to parameter extraction from IV measurements of Pt/Cr_2O_3:Mg/β-Ga_2O_3 devices, we demonstrate that AI-driven laboratory automation can quickly yield insights into semiconductor device physics.

The AI-engineered control module, along with the user-friendly graphical user interface and the parameter extraction algorithm described in this paper, are made open-source through GitHub, so that the whole community can benefit from and contribute to the further development of these tools. Looking ahead, this LMM-based software development method holds the potential to revolutionize research automation by enabling the control of a diverse spectrum of laboratory instruments. Potential future applications include but are not limited to multimeters, temperature controllers, power supplies, mass flow controllers, programmable logic controllers, or any other equipment that features standard communication and command protocols such as Modbus, OPC UA, SCPI. The resulting increased laboratory automation would pave the way for a more interconnected and efficient research environment through the use of large language models.

Data availability

Source code for the ChatGPT-crafted control module and GUI is available at: **https://github.com/NREL/Keithley_GPT**. The differential evolution algorithm source code and Jupyter notebooks used to analyze the results and generate the figures in this paper are available at: **https://github.com/NREL/DE**. The prompt templates, JSON file, and the resulting Python files created using these templates are available in the ESI.†

Author contributions

D. Febba and Andriy Zakutayev conceptualized the system described in this work. D. Febba drafted the manuscript with input from all the co-authors, and operated the characterization system. K. Egbo fabricated the heterojunction devices. D. Febba and W. Callahan tested the probe station atmosphere control and scripting routines. ChatGPT (GPT-4 version) was used to craft Python code and to refine the text. The final version of this manuscript was approved by all authors.

Conflicts of interest

There are no conflicts to declare.

Acknowledgements

This work was authored by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. Funding is provided by the Office of Energy Efficiency and Renewable Energy (EERE) Advanced Materials & Manufacturing Technologies Office (AMMTO) (IV parameter extraction and semiconductor device measurements); and by a Laboratory Directed Research and Development (LDRD) program at NREL (AI-enhanced development of instrument control and graphical user interface). The views expressed in the article do not necessarily represent the views of the DOE or the U.S. Government.

Notes and references

1 A. D. White, G. M. Hocky, H. A. Gandhi, M. Ansari, S. Cox, G. P. Wellawatte, S. Sasmal, Z. Yang, K. Liu, Y. Singh and W. J. Peña Ccoa, *Digital Discovery*, 2023, **2**, 368–376.
2 C. M. Castro Nascimento and A. S. Pimentel, *J. Chem. Inf. Model.*, 2023, **63**, 1649–1655.
3 M. Thway, A. K. Y. Low, S. Khetan, H. Dai, J. Recatala-Gomez, A. P. Chen and K. Hippalgaonkar, *Digital Discovery*, 2024, **3**, 328–336.

© 2025 The Author(s). Published by the Royal Society of Chemistry

Open Access Article. Published on 23 October 2024. Downloaded on 1/16/2025 5:45:16 PM. This article is licensed under a Creative Commons Attribution 3.0 Unported Licence.

4 K. G. Yager, *Digital Discovery*, 2023, **2**, 1850–1861.
5 D. A. Boiko, R. MacKnight, B. Kline and G. Gomes, *Nature*, 2023, **624**, 570–578.
6 R. S. Aal E Ali, J. Meng, M. E. I. Khan and X. Jiang, *Artif. Intell. Chem.*, 2024, **2**, 100049.
7 R. Van Noorden and R. Webb, *Nature*, 2023, **624**, 509.
8 K. M. Jablonka, Q. Ai, A. Al-Feghali, S. Badhwar, J. D. Bocarsly, A. M. Bran, S. Bringuier, L. C. Brinson, K. Choudhary, D. Circi, S. Cox, W. A. de Jong, M. L. Evans, N. Gastellu, J. Genzling, M. V. Gil, A. K. Gupta, Z. Hong, A. Imran, S. Kruschwitz, A. Labarre, J. Lála, T. Liu, S. Ma, S. Majumdar, G. W. Merz, N. Moitessier, E. Moubarak, B. Mouriño, B. Pelkie, M. Pieler, M. C. Ramos, B. Ranković, S. G. Rodriques, J. N. Sanders, P. Schwaller, M. Schwarting, J. Shi, B. Smit, B. E. Smith, J. Van Herck, C. Völker, L. Ward, S. Warren, B. Weiser, S. Zhang, X. Zhang, G. A. Zia, A. Scourtas, K. J. Schmidt, I. Foster, A. D. White and B. Blaiszik, *Digital Discovery*, 2023, **2**(5), 1233–1250.
9 K. Emery, in *Measurement and Characterization of Solar Cells and Modules*, John Wiley & Sons, Ltd, 2010, ch. 18, pp. 797–840.
10 R. C. Kurchin, J. R. Poindexter, V. Vähänissi, H. Savin, C. del Cañizo and T. Buonassisi, *IEEE J. Photovoltaics*, 2020, **10**, 1532–1537.
11 R. O. Ocaya, A. A. Akinyelu, A. G. Al-Sehemi, A. Dere, A. A. Al-Ghamdi and F. Yakuphanoğlu, *Sci. Rep.*, 2023, **13**, 13990.
12 D. M. Fébba, R. M. Rubinger, A. F. Oliveira and E. C. Bortoni, *Sol. Energy*, 2018, **174**, 628–639.
13 C. Valdivieso, R. Rodriguez, A. Crespo-Yepes, J. Martin-Martinez and M. Nafria, *Solid-State Electron.*, 2023, **209**, 108759.
14 H. Zhai, Z. Wu and Z. Fang, *Ceram. Int.*, 2022, **48**, 24213–24233.
15 H. Y. Wong, M. Xiao, B. Wang, Y. K. Chiu, X. Yan, J. Ma, K. Sasaki, H. Wang and Y. Zhang, *IEEE J. Electron Devices Soc.*, 2020, **8**, 992–1000.
16 S. Li, W. Gong and Q. Gu, *Renewable Sustainable Energy Rev.*, 2021, **141**, 110828.
17 H. Jin, L. Huang, H. Cai, J. Yan, B. Li and H. Chen, From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future, *arXiv*, 2024, preprint, arXiv:2408.02479, DOI: **10.48550/arXiv.2408.02479**, **https://arxiv.org/abs/2408.02479**.
18 J. Liu, K. Wang, Y. Chen, X. Peng, Z. Chen, L. Zhang and Y. Lou, Large Language Model-Based Agents for Software Engineering: A Survey, *arXiv*, 2024, preprint, arXiv:2409.02977, DOI: **10.48550/arXiv.2409.02977**, **https://arxiv.org/abs/2409.02977**.
19 Y. Wang, W. Zhong, Y. Huang, E. Shi, M. Yang, J. Chen, H. Li, Y. Ma, Q. Wang and Z. Zheng, Agents in Software Engineering: Survey, Landscape, and Vision, *arXiv*, 2024, preprint, arXiv:2409.09030, DOI: **10.48550/arXiv.2409.09030**, **https://arxiv.org/abs/2409.09030**.
20 D. M. Fébba, K. R. Talley, K. Johnson, S. Schaefer, S. R. Bauers, J. S. Mangum, R. W. Smaha and A. Zakutayev, *APL Mater.*, 2023, **11**, 071119.
21 M. Renze and E. Guven, Self-Reflection in LLM Agents: Effects on Problem-Solving Performance, *arXiv*, 2024, preprint, arXiv:2405.06682, DOI: **10.48550/arXiv.2405.06682**, **https://arxiv.org/abs/2405.06682**.
22 Cursor, **https://www.cursor.com/**, accessed October 2024.
23 LangChain, **https://www.langchain.com/**, accessed October 2024.
24 Devin, **https://www.devin.com/**, accessed October 2024.
25 Amazon Q, **https://aws.amazon.com/q/**, accessed October 2024.
26 J. Brest, S. Greiner, B. Boskovic, M. Mernik and V. Zumer, *IEEE Trans. Evol. Comput.*, 2006, **10**, 646–657.
27 D. M. Fébba, E. C. Bortoni, A. F. Oliveira and R. M. Rubinger, *Sol. Energy*, 2020, **201**, 420–436.
28 S. K. Lam, A. Pitrou and S. Seibert, *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, New York, NY, USA, 2015.
29 C. Chellaswamy and R. Ramesh, *Renewable Energy*, 2016, **97**, 823–837.
30 P. Lin, S. Cheng, W. Yeh, Z. Chen and L. Wu, *Sol. Energy*, 2017, **144**, 594–603.
31 F. Biscani and D. Izzo, *J. Open Source Softw.*, 2020, **5**, 2338.
32 D. Fébba, V. Paratte, L. Antognini, J. Dréon, J. Hurni, J. Thomet, R. Rubinger, E. Bortoni, C. Ballif and M. Boccard, *IEEE J. Photovoltaics*, 2021, **11**, 1350–1357.
33 W. Jung and M. Guziewicz, *Mater. Sci. Eng., B*, 2009, **165**, 57–59.
34 M. Ćalasan, S. H. Abdel Aleem and A. F. Zobaa, *Energy Convers. Manage.*, 2020, **210**, 112716.
35 S. Aazou, M. S. White, M. Kaltenbrunner, Z. Sekkat, D. A. M. Egbe and E. M. Assaid, *Energies*, 2022, **15**, 1667.
36 L. Lóczi, *Appl. Math. Comput.*, 2022, **433**, 127406.
37 W. A. Callahan, E. Supple, D. Ginley, M. Sanders, B. P. Gorman, R. O'Hayre and A. Zakutayev, *J. Vac. Sci. Technol., A*, 2023, **41**, 043211.
38 W. A. Callahan, K. Egbo, C.-W. Lee, D. Ginley, R. O'Hayre and A. Zakutayev, *Appl. Phys. Lett.*, 2024, **124**, 153504.
39 Y. Yao, R. Gangireddy, J. Kim, K. K. Das, R. F. Davis and L. M. Porter, *J. Vac. Sci. Technol. B*, 2017, **35**, 03D113.
40 K. Heinselman, P. Walker, A. Norman, P. Parilla, D. Ginley and A. Zakutayev, *J. Vac. Sci. Technol., A*, 2021, **39**, 040402.
41 S. H. Sohel, R. Kotecha, I. S. Khan, K. N. Heinselman, S. Narumanchi, M. B. Tellekamp and A. Zakutayev, *Phys. Status Solidi A*, 2023, 2300535.
42 C.-T. Lee and J.-T. Yan, *Sens. Actuators, B*, 2010, **147**, 723–729.

© 2025 The Author(s). Published by the Royal Society of Chemistry