Interoperable Natural Language Interfaces for Self-Driving Labs via Model Context Protocol

Anonymous Author(s)
Affiliation
Address
email

Abstract

The development of self-driving laboratories (SDLs) is accelerating materials discovery by automating synthesis and characterization with robotic platforms and diverse instrumentation. However, the lack of standardized software interfaces hinders their broader adoption and interoperability. A parallel challenge arises with building agentic AI that can reliably control diverse physical systems. In this work, we present an architecture that integrates the Model Context Protocol (MCP)—an open standard for tool interaction with large language models (LLMs)—with an interoperable laboratory orchestration layer. This enables natural language interaction across the full spectrum of SDL functionality, from direct instrument control to closed-loop workflow design. We demonstrate its capabilities through two representative use cases: (1) optimization of liquid handling accuracy and (2) synthesis with computer vision monitoring. By bridging natural language interfaces, standardized protocols, and SDL interoperability, this architecture lowers the barrier to entry for both domain scientists and developers, paving the way for more adoptable, scalable, and intelligent laboratory automation.

1 Introduction

The convergence of artificial intelligence (AI) and automated experimentation is catalyzing a paradigm shift in materials science [1]. Generative models can now propose novel materials *in silico*, promising to unlock transformative innovations in critical areas such as energy, healthcare, and sustainability [2, 3]. The ultimate goal is to realize this vision in practice through self-driving laboratories (SDLs), which integrate AI with robots to automate synthesis and characterization [4, 5]. In the past decade, multiple state-of-the-art SDLs have been developed to address material, organic chemistry, and global challenges in health, climate, and energy [6, 7, 8, 9, 10, 11, 12].

Despite these advances, the necessity of custom scripts and nature of diverse hardware configurations require considerable programming proficiency and often results in isolated, non-transferable systems. The lack of standardized interfaces has also hindered the generalization of these approaches, slowing down adoption within the broader scientific community. Several general-purpose graphical orchestration platforms have been developed, including AlabOS [13], ChemOS 2.0 [14], NIMS-OS [15] and IvoryOS [16]. At the same time, the rapid advancement of Large Language Models (LLMs) enables protocol translation and robotic action planning, from experimental description to machine-executable code [17, 18, 19, 20, 21, 22]. The capability of controlling a robotic system through natural language communication promises more intuitive human–robot interaction and scalable multi-agent integration [4, 23, 24, 25, 26, 27].

To bridge the gap between high-level scientific intent and low-level hardware control, we introduce a Model Context Protocol (MCP) server that enables natural language interaction with any Python-based SDLs. The framework extends IvoryOS [16] with the MCP open standard, creating a robust

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

and intuitive conversational interface. We demonstrate how this architecture democratizes laboratory automation by allowing users to perform direct hardware control, design complex workflows, and manage data through simple natural language commands. With two distinct use cases — one focused on autonomous optimization and the other on a multi-instrument workflow — we showcase the system's ability to translate complex scientific goals into executable, production-ready code.

2 Background: The IvoryOS Foundation

IvoryOS is an open-source orchestrator that automatically generates web interfaces for Python-based SDLs [16]. The software features three design principles: (i) automatic discovery of available functions and instruments, (ii) a visual programming interface to lower the entry barrier for domain experts, and (iii) integration of optimizers for adaptive experimentation (Figure 1). This allows researchers, even those with limited programming experience, to design, manage, and execute complex experiments. The foundational paper demonstrated its successful integration across six different SDLs, proving its adaptability.

3 System Architecture

Extending the base architecture, the natural language interface is achieved by creating an MCP server that acts as a bridge between the high-level reasoning capabilities of LLMs and the low-level control functions managed by IvoryOS (Figure 1). **IvoryOS Backend:** IvoryOS serves as the foundational layer, providing a robust interface to diverse laboratory hardware and tasks. Its key feature is the ability to introspect existing Python scripts, automatically discovering functions that control instruments (e.g., pumps, robotic arms, sensors) and exposing them through a web API. This eliminates the need for manual API creation and allows scientists to use their existing control code. **IvoryOS Client:** The IvoryOS Python client is a convenient interface to interact with the IvoryOS API, providing explicit functions for direct control, workflow configuration, execution, status queries, and data retrieval (Appendix A). **MCP Server:** The MCP server exposes IvoryOS client functions into a set of "tools" that are discoverable and callable by an LLM agent. When an agent queries the MCP server, it receives a manifest of all available tools, including their names, descriptions, and required parameters. This allows the agent to understand the lab's capabilities without any prior hard-coding. **LLM:** The MCP host is model-agnostic. For simplicity and ease of adoption, we demonstrate the integration using the Claude Desktop App with the Claude Sonnet 4 model.

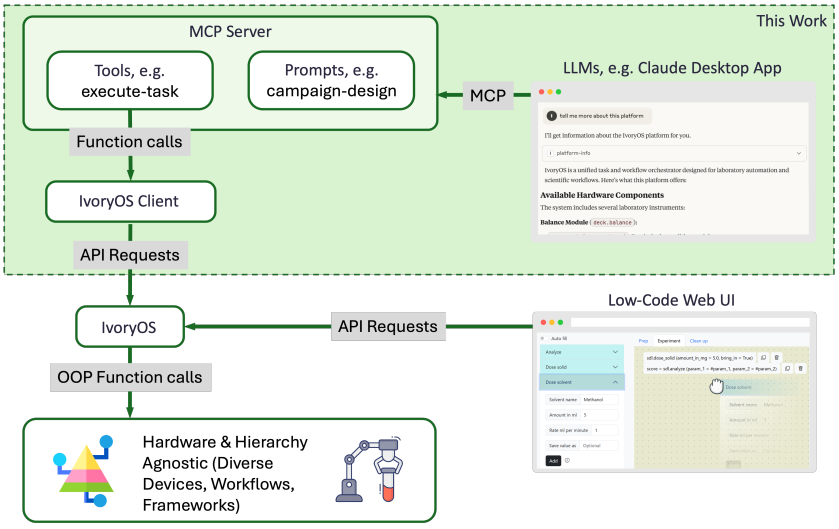

Figure 1: **The IvoryOS MCP System Architecture.** Our contribution (top, green area) is an MCP server that translates natural language commands into function calls. These are processed by the IvoryOS Client and sent as API requests to the underlying IvoryOS platform.

4 Demonstrations in SDL Context

4.1 Base Case: Direct Hardware Control

Direct control of platform components is achieved via the `execute-task` tool, which enables the LLM to call any single function of the SDL. This granular control can perform immediate actions without the overhead of creating a formal script. For instance, a **charge solid** operation is scripted as the following function definition. When prompting for operation, the task can be triggered through `execute-task` function calling handled by the LLM.

Example Operation

```
class Protocol:
    def charge_solid(
        self,
        solid_weight: float,
        container_type: str,
        index: str = None
    ):...
```

User Prompt

Charge 5 mg of solid to HPLC vial A1.

Generated Tool Call

```
execute_task(
  component="deck.protocol",
  method="charge_solid",
  kwargs={
    "solid_weight": 5.0,
    "container_type": "hplc",
    "index": "A1"
  }
)
```

4.2 Use Case 1: Workflow Optimization

Precise control of liquid handling is fundamental in chemistry and materials synthesis. We tasked the system to optimize the dosing accuracy of a mobile liquid handler, a common and often challenging task. The workflows of **charge solvent** and **evaluate mean error** are predefined on the robotic platform designed for liquid-liquid extraction. An LLM agent was given the high-level goal: optimize the mobile liquid handler accuracy in 60 trials with the following parameters (Figure 2A). The agent autonomously designed and executed an optimization workflow. It first used the 'platform-info' tool to understand the platform, such as available hardware and execution rules, followed by loading existing workflows or scripting from scratch. The closed-loop process is designed with the help of an included prompt for inputting parameters and objectives for **ax-platform** [28] (Appendix B). All tool calling decisions are driven entirely by the LLM, demonstrating the system's ability to perform hardware control and execute complex optimization routines through natural language.

4.3 Use Case 2: Multi-Equipment Coordination and Process Control

Building on the optimization of existing workflows demonstrated in Use Case 1, we present a more complex scenario: a *de novo* monitoring process with computer vision. The system was given the high-level natural language specification: Implement continuous stirred tank reactors (CSTR) using computer vision with detailed control logic (Figure 2B). The LLM agent autonomously generated a sophisticated workflow, coordinating nine different instruments: two computer vision sensors for real-time volume monitoring, two syringe pumps for slurry transfer, three flow pumps for reagents,

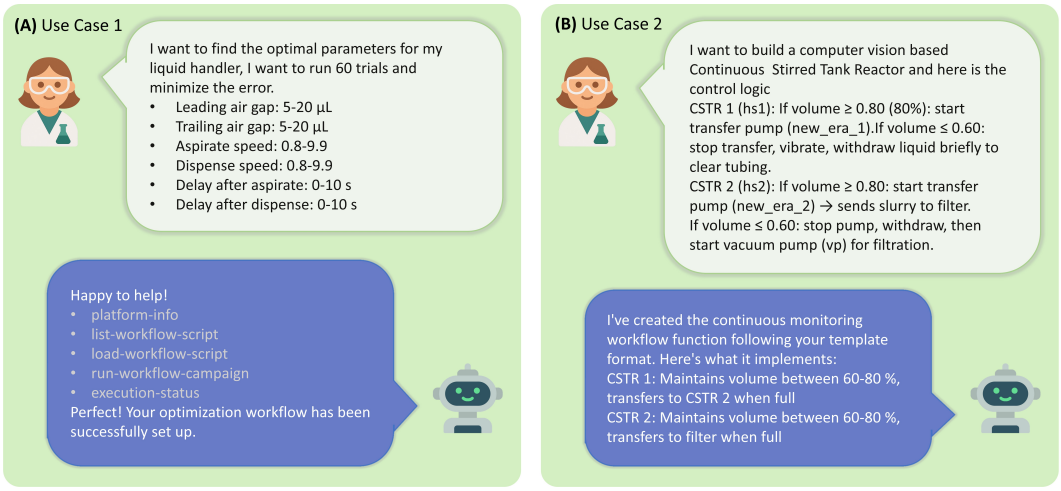

Figure 2: **Exemplary Use Cases of the MCP System. (A) Use Case 1: Autonomous Optimization.** A user specifies a high-level optimization goal for a liquid handler, including the parameters and their respective ranges. The LLM agent interprets this request, selects the necessary tools, and initiates an optimization campaign using the `run-workflow-campaign` feature. **(B) Use Case 2: Complex Workflow Synthesis from Natural Language.** A user provides a detailed specification for a multi-instrument continuous synthesis process. The LLM agent parses this complex logic and generate an executable workflow, demonstrating its capability for de novo process design and implementation.

a vacuum pump for filtration, and a vibrator for anti-clogging maintenance. The resulting code implements a complex state-machine logic where CSTR-1 transfers supersaturated solution to CSTR-2 when its volume exceeds 80%, while CSTR-2 transfers its contents to a filter. Crucially, the LLM translated nuanced natural language safety protocols—such as "stops vacuum pumps before transfers" and "implements a back-flush to prevent clogging"—directly into instrument commands (Appendix C).

This use case demonstrates the system's ability to translate a complex process description into production-ready automation code, complete with sophisticated orchestration, real-time feedback control, and robust error handling. This capability is essential for the future of autonomous materials discovery, where such complex experimental protocols must be generated and executed reliably at scale.

5 Conclusion and Future Work

In summary, we have shown that integrating MCP with a standardized laboratory orchestration layer enables natural language interfaces to span the full spectrum of SDL capabilities, ranging from instrument-level control to closed-loop workflow optimization. The two representative use cases—liquid-handling optimization and synthesis with vision-based monitoring—illustrate both the versatility and practicality of this approach. Looking ahead, we envision this architecture serving as a foundation for broader interoperability across diverse laboratory platforms, lowering adoption barriers for researchers, and facilitating the integration of increasingly autonomous, agentic AI systems in material science.

References

[1] Milad Abolhasani, Keith A. Brown, and Guest Editors. "Role of AI in experimental materials science". In: *MRS Bulletin* 48.2 (Feb. 2023), pp. 134–141. ISSN: 1938-1425. DOI: 10.1557/s43577-023-00482-y. URL: https://doi.org/10.1557/s43577-023-00482-y.

[2] Albertus Denny Handoko and Riko I Made. *Artificial Intelligence and Generative Models for Materials Discovery – A Review*. 2025. arXiv: 2508.03278 [cond-mat.mtrl-sci]. URL: https://arxiv.org/abs/2508.03278.

[3] Mehrad Ansari et al. *dZiner: Rational Inverse Design of Materials with AI Agents*. 2024. arXiv: 2410.03963 [physics.chem-ph]. URL: https://arxiv.org/abs/2410.03963.

[4] Gary Tom et al. "Self-Driving Laboratories for Chemistry and Materials Science". en. In: *Chemical Reviews* (Aug. 2024), acs.chemrev.4c00055. ISSN: 0009-2665, 1520-6890. DOI: 10.1021/acs.chemrev.4c00055. URL: https://pubs.acs.org/doi/10.1021/acs.chemrev.4c00055 (visited on 08/22/2024).

[5] Richard B. Canty et al. "Science acceleration and accessibility with self-driving labs". en. In: *Nature Communications* 16.1 (Apr. 2025), p. 3856. ISSN: 2041-1723. DOI: 10.1038/s41467-025-59231-1. URL: https://www.nature.com/articles/s41467-025-59231-1 (visited on 06/21/2025).

[6] Yixiang Ruan et al. "An automatic end-to-end chemical synthesis development platform powered by large language models". en. In: *Nature Communications* 15.1 (Nov. 2024), p. 10160. ISSN: 2041-1723. DOI: 10.1038/s41467-024-54457-x. URL: https://www.nature.com/articles/s41467-024-54457-x (visited on 06/21/2025).

[7] B. P. MacLeod et al. "Self-driving laboratory for accelerated discovery of thin-film materials". en. In: *Science Advances* 6.20 (May 2020), eaaz8867. ISSN: 2375-2548. DOI: 10.1126/sciadv.aaz8867. URL: https://www.science.org/doi/10.1126/sciadv.aaz8867 (visited on 05/08/2024).

[8] Liam Roberts et al. "Automating stochastic antibody–drug conjugation: a self-driving lab approach for enhanced therapeutic development". en. In: *Digital Discovery* (2025), 10.1039.D4DD00363B. ISSN: 2635-098X. DOI: 10.1039/D4DD00363B. URL: https://xlink.rsc.org/?DOI=D4DD00363B (visited on 04/03/2025).

[9] Connor C. Rupnow et al. "A self-driving laboratory optimizes a scalable process for making functional coatings". en. In: *Cell Reports Physical Science* 4.5 (May 2023), p. 101411. ISSN: 26663864. DOI: 10.1016/j.xcrp.2023.101411. URL: https://linkinghub.elsevier.com/retrieve/pii/S2666386423001856 (visited on 07/17/2024).

[10] S. Hessam M. Mehr et al. "A universal system for digitization and automatic execution of the chemical synthesis literature". en. In: *Science* 370.6512 (Oct. 2020), pp. 101–108. ISSN: 0036-8075, 1095-9203. DOI: 10.1126/science.abc2986. URL: https://www.science.org/doi/10.1126/science.abc2986 (visited on 03/14/2024).

[11] Kazunori Nishio et al. "A digital laboratory with a modular measurement system and standardized data format". en. In: *Digital Discovery* (2025), 10.1039.D4DD00326H. ISSN: 2635-098X. DOI: 10.1039/D4DD00326H. URL: https://xlink.rsc.org/?DOI=D4DD00326H (visited on 06/21/2025).

[12] Chengshi Wang et al. "Autonomous platform for solution processing of electronic polymers". en. In: *Nature Communications* 16.1 (Feb. 2025), p. 1498. ISSN: 2041-1723. DOI: 10.1038/s41467-024-55655-3. URL: https://www.nature.com/articles/s41467-024-55655-3 (visited on 06/21/2025).

[13] Yuxing Fei et al. "AlabOS: a Python-based reconfigurable workflow management framework for autonomous laboratories". en. In: *Digital Discovery* 3.11 (2024), pp. 2275–2288. ISSN: 2635-098X. DOI: 10.1039/D4DD00129J. URL: https://xlink.rsc.org/?DOI=D4DD00129J (visited on 04/07/2025).

[14] Malcolm Sim et al. "ChemOS 2.0: An orchestration architecture for chemical self-driving laboratories". en. In: *Matter* 7.9 (Sept. 2024), pp. 2959–2977. ISSN: 25902385. DOI: 10.1016/j.matt.2024.04.022. URL: https://linkinghub.elsevier.com/retrieve/pii/S2590238524001954 (visited on 09/27/2024).

[15] Ryo Tamura, Koji Tsuda, and Shoichi Matsuda. "NIMS-OS: an automation software to implement a closed loop between artificial intelligence and robotic experiments in materials science". en. In: *Science and Technology of Advanced Materials: Methods* 3.1 (Dec. 2023), p. 2232297. ISSN: 2766-0400. DOI: 10.1080/27660400.2023.2232297. URL: https://www.tandfonline.com/doi/full/10.1080/27660400.2023.2232297 (visited on 06/21/2025).

[16] Wenyu Zhang et al. "IvoryOS: an interoperable web interface for orchestrating Python-based self-driving laboratories". en. In: *Nature Communications* 16.1 (June 2025), p. 5182. ISSN: 2041-1723. DOI: 10.1038/s41467-025-60514-w. URL: https://www.nature.com/articles/s41467-025-60514-w (visited on 06/21/2025).

[17] Naruki Yoshikawa et al. "Large language models for chemistry robotics". en. In: *Autonomous Robots* 47.8 (Dec. 2023), pp. 1057–1086. ISSN: 0929-5593, 1573-7527. DOI: 10.1007/s10514-023-10136-2. URL: https://link.springer.com/10.1007/s10514-023-10136-2 (visited on 03/14/2024).

[18] Kourosh Darvish et al. "ORGANA: A Robotic Assistant for Automated Chemistry Experimentation and Characterization". en. In: *arXiv* (Jan. 2024). arXiv:2401.06949 [cs], arXiv:2401.06949. URL: http://arxiv.org/abs/2401.06949 (visited on 04/23/2024).

[19] Wenyu Zhang et al. "Leveraging GPT-4 to transform chemistry from paper to practice". en. In: *Digital Discovery* (2024). ISSN: 2635-098X. DOI: 10.1039/D4DD00248B. URL: https://xlink.rsc.org/?DOI=D4DD00248B (visited on 10/19/2024).

[20] Mayk Caldas Ramos, Christopher J. Collison, and Andrew D. White. "A review of large language models and autonomous agents in chemistry". en. In: *Chemical Science* 16.6 (2025), pp. 2514–2572. ISSN: 2041-6520, 2041-6539. DOI: 10.1039/D4SC03921A. URL: https://xlink.rsc.org/?DOI=D4SC03921A (visited on 06/21/2025).

[21] Kan Hatakeyama-Sato et al. *Perspective on Utilizing Foundation Models for Laboratory Automation in Materials Research*. 2025. arXiv: 2506.12312 [cs.RO]. URL: https://arxiv.org/abs/2506.12312.

[22] Bastian Ruehle. "Natural language processing for automated workflow and knowledge graph generation in self-driving labs". In: *Digital Discovery* 4.6 (2025). Publisher: RSC, pp. 1534–1543. DOI: 10.1039/D5DD00063G. URL: http://dx.doi.org/10.1039/D5DD00063G.

[23] Tianshi Zheng et al. *From Automation to Autonomy: A Survey on Large Language Models in Scientific Discovery*. en. arXiv:2505.13259 [cs]. May 2025. DOI: 10.48550/arXiv.2505.13259. URL: http://arxiv.org/abs/2505.13259 (visited on 06/21/2025).

[24] Daniil A. Boiko et al. "Autonomous chemical research with large language models". en. In: *Nature* 624.7992 (Dec. 2023), pp. 570–578. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/s41586-023-06792-0. URL: https://www.nature.com/articles/s41586-023-06792-0 (visited on 03/14/2024).

[25] Shuxiang Cao et al. *Agents for self-driving laboratories applied to quantum computing*. en. arXiv:2412.07978 [cs]. June 2025. DOI: 10.48550/arXiv.2412.07978. URL: http://arxiv.org/abs/2412.07978 (visited on 06/21/2025).

[26] Yunheng Zou et al. *El Agente: An Autonomous Agent for Quantum Chemistry*. en. arXiv:2505.02484 [cs]. May 2025. DOI: 10.48550/arXiv.2505.02484. URL: http://arxiv.org/abs/2505.02484 (visited on 06/21/2025).

[27] Juraj Gottweis et al. *Towards an AI co-scientist*. 2025. arXiv: 2502.18864 [cs.AI]. URL: https://arxiv.org/abs/2502.18864.

[28] *Adaptive Experimentation Platform*. URL: https://ax.dev/.

A Summary of MCP Server Tools

Table 1: Summary of MCP Server Tools. The server exposes a comprehensive set of tools for interacting with the IvoryOS platform, categorized by function.

Category	Feature	Description
General Tools	`platform-info`	Get platform info.
	`execution-status`	Check system status and last task outcome.
Workflow Design	`list-workflow-scripts`	List all available workflow scripts from the database.
	`load-workflow-script`	Load a specific workflow script for execution.
	`submit-workflow-script`	Save a new or modified workflow script to the database.
Workflow Data	`list-workflow-data`	List data from previous workflow executions.
	`load-workflow-data`	Load a specific execution log.
Direct Control	`execute-task`	Directly call a single platform or instrument function.
Workflow Execution	`run-workflow-repeat`	Run with static parameters.
	`run-workflow-kwargs`	Run with dynamic parameters.
	`run-workflow-campaign`	Run with optimization campaign configs
Workflow Control	`pause-and-resume`	Pause or resume an ongoing workflow execution.
	`abort-pending-workflow`	Finish the current iteration and abort subsequent runs.
	`stop-current-workflow`	Safely stop after the current step.

B Use Case 1 Campaign Parameters

The following listing shows the full set of parameters translated by the LLM from the user's natural language prompt and used to initiate the `run-workflow-campaign` tool for the liquid handler optimization task.

```
{
  "repeat": 60,
  "objectives": [
    {
      "name": "optimization_result",
      "minimize": true,
      "threshold": null
    }
  ],
  "parameters": [
    {
      "name": "leading_air_gap_ul",
      "type": "range",
      "bounds": [5, 20],
      "value_type": "float"
    },
    {
      "name": "trailing_air_gap_ul",
      "type": "range",
      "bounds": [5, 20],
      "value_type": "float"
    },
    {
      "name": "aspirate_speed",
      "type": "range",
      "bounds": [0.8, 9.9],
      "value_type": "float"
    },
    {
      "name": "dispense_speed",
      "type": "range",
      "bounds": [0.8, 9.9],
      "value_type": "float"
    },
    {
      "name": "delay_after_aspirate",
      "type": "range",
      "bounds": [0, 10],
      "value_type": "float"
    },
    {
      "name": "delay_after_dispense",
      "type": "range",
      "bounds": [0, 10],
      "value_type": "float"
    }
  ]
}
```

Listing 1: Parameters for Use Case 1

C Use Case 2 Generated Workflow Code

The following listing is the full Python workflow generated by the LLM agent from the natural language specification for the continuous crystallization process. This code demonstrates the agent's ability to synthesize complex, multi-instrument logic, including state management and safety protocols.

```python

def workflow_static():
    print("Starting CSTR control loop...")

    # Step 1 - Startup: Check initial pump statuses
    pump1_status = deck.new_era_1.get_status()
    pump2_status = deck.new_era_2.get_status()
    print(f"New Era pump 1 initial status: {pump1_status}")
    print(f"New Era pump 2 initial status: {pump2_status}")

    # Initialize variables to track pump states
    pump1_dispensing = False
    pump2_dispensing = False

    # Main control loop
    while True:
        # Step 2 - Vision Monitoring: Read liquid levels
        hs1_volume_1 = deck.heinsight_1.volume_1()
        hs2_volume_1 = deck.heinsight_2.volume_1()

        print(f"CSTR 1 volume: {hs1_volume_1}")
        print(f"CSTR 2 volume: {hs2_volume_1}")

        # Step 3 - Control of CSTR 1 (new_era_1 logic)
        if hs1_volume_1 is not None:
            # Case A: Reactor full
            if hs1_volume_1 >= 0.80 and not pump1_dispensing:
                print("CSTR 1 full - starting dispensing to CSTR 2")
                deck.new_era_1.set_direction_dipense()
                deck.new_era_1.set_rate(98.0)
                deck.new_era_1.start()

                # Start side pump sf10_3 for balance
                deck.sf10_3.connect()
                deck.sf10_3.set_mode("FLOW")
                deck.sf10_3.set_flow_rate(0.05)
                deck.sf10_3.start()

                pump1_dispensing = True
                print("New Era 1 dispensing at 98 mL/min, SF10_3
                    started at 0.05 flow rate")

            # Case B: Reactor low
            elif hs1_volume_1 <= 0.60 and pump1_dispensing:
                print("CSTR 1 low - stopping and cleaning")
                deck.new_era_1.stop()

                # Activate vibrator to prevent clogging
                deck.vibrator.start()
                time.sleep(2.5)
                deck.vibrator.stop()
                print("Vibrator activated for 2.5 seconds")

                # Back-flush tubing
                deck.new_era_1.set_direction_withdraw()
                deck.new_era_1.set_rate(98.0)
                deck.new_era_1.start()
```

```
                time.sleep(5.0)
                deck.new_era_1.stop()

                pump1_dispensing = False
                print("New Era 1 back-flushed and stopped")

        # Step 4 - Control of CSTR 2 (new_era_2 logic)
        if hs2_volume_1 is not None:
            # Case A: Reactor full
            if hs2_volume_1 >= 0.80 and not pump2_dispensing and not
                pump1_dispensing:
                print("CSTR 2 full - waiting buffer time then
                    transferring to filter")
                time.sleep(100.0)  # Buffer wait

                # Stop vacuum pump before transfer
                deck.vp.stop()
                print("Vacuum pump stopped")

                # Start dispensing to filter
                deck.new_era_2.set_direction_dipense()
                deck.new_era_2.set_rate(98.0)
                deck.new_era_2.start()

                pump2_dispensing = True
                print("New Era 2 dispensing to filter at 98 mL/min")

            # Case B: Reactor low
            elif hs2_volume_1 <= 0.60 and pump2_dispensing:
                print("CSTR 2 low - stopping and cleaning")
                deck.new_era_2.stop()

                # Clear tubing with withdraw
                deck.new_era_2.set_direction_withdraw()
                deck.new_era_2.set_rate(98.0)
                deck.new_era_2.start()
                time.sleep(5.0)
                deck.new_era_2.stop()

                # Start vacuum pump for filtration
                deck.vp.start()
                time.sleep(5.0)

                pump2_dispensing = False
                print("New Era 2 cleaned, vacuum pump started for
                    filtration")

        # Step 5 - Loop delay
        time.sleep(1.0)

    return {
        'final_cstr1_volume': hs1_volume_1,
        'final_cstr2_volume': hs2_volume_1,
        'pump1_final_status': pump1_dispensing,
        'pump2_final_status': pump2_dispensing
    }
```

Listing 2: Generated workflow for Use Case 2