NIMO Controller: An accessible self-driving laboratory orchestrator based on the model context protocol

Naruki Yoshikawa [1] **Ryo Tamura** [1 2]

[1]*National Institute for Materials Science (NIMS), Tsukuba, Ibaraki, Japan* [2]*The University of Tokyo, Kashiwa, Chiba, Japan*. Correspondence to: Naruki Yoshikawa yoshikawa.naruki@nims.go.jp.

1. Introduction

Self-driving laboratories (SDLs) combine robotic automated experiments with data-driven experimental design to accelerate scientific discovery [1]. Despite growing interest, practical SDL deployment still faces challenges. Among them, software development poses an accessibility problem: most SDL software requires programming expertise, limiting adoption by domain experts who design experiments but do not write code. To address this issue, orchestration software has been proposed to support the development of SDL software frameworks. We have previously reported the integration of IvoryOS [2] and NIMO [3] to provide a graphical user interface for laboratory workflow design incorporating algorithmic decision making [4]. Although IvoryOS automatically generates an accessible interface from Python code, SDLs often require integration of legacy devices that may not support Python [5]. In addition, given the growing interest in AI scientists, an SDL interface designed for AI systems is necessary.

To provide interoperability between large language models (LLMs) and external tools, the model context protocol (MCP) has recently been proposed. MCP interfaces enable seamless connections between external tools or data resources and LLMs. This paper introduces an MCP-centric design of an SDL abstraction layer and describes orchestration software based on this design. The contributions of this paper are as follows:

Hardware abstraction through MCP: We adopt MCP to wrap each function of the SDL and achieve loose coupling between orchestration software and laboratory hardware. This design enables language-agnostic development on the hardware side and plug-and-play extensibility, where adding a new device requires only publishing a new MCP server, without any changes to the client code.

Visual programming interface: A visual block-based editor based on Blockly is introduced to provide an accessible interface for end users. The blocks are automatically generated through the tool discovery function of MCP, and users can design experimental workflows with decision-making by NIMO.

Natural language interface: Users can invoke a chat-based agent to access the SDL through MCP. They can choose between two interfaces: manual workflow design or automated execution through an AI agent. The agent supports human-in-the-loop tool invocation, allowing users to individually approve or reject tool execution, thereby providing a fine-grained safety mechanism.

2. System Architecture

2.1 Overview

NIMO Controller acts as an MCP host that sits between the user and MCP servers (Figure 1). Users can access the frontend interface of the NIMO Controller, which integrates both a block-based interface and a natural language interface into a single web page, through a web browser. The NIMO Controller communicates with two backend MCP servers: one for NIMO which provides the decision-making engine, and one for the SDL server, which provides access to laboratory hardware.

2.2 User interface

2.2.1 Visual programming interface

This interface uses Blockly to provide a drag-and-drop workflow editor (Figure 2). Three different toolboxes of blocks are provided: core, NIMO, and lab. The core toolbox provides blocks for controlling the workflow, such as a repeat block. The NIMO toolbox provides blocks to access NIMO's decision-making functionalities, which are described in Section 2.3.1. The lab toolbox provides blocks to execute SDL functions and these blocks are automatically generated from information obtained from the backend MCP server of the SDL. At page load, the frontend queries the capabilities of the MCP servers obtained during the MCP initialization stage. For each discovered tool, a Blockly block definition whose input fields (e.g., numbers and text fields) are derived from the tool definition is dynamically generated.

Users can compose workflows by snapping blocks together. When the user clicks the run button, the designed workflow is sent to the backend and executed by invoking the MCP server's functionalities. The interface highlights the currently executing block in real time.

2.2.2 Natural language interface

Users can type natural-language instructions into a chat interface (Figure 3). The orchestrator forwards messages to an LLM agent that has access to the backend MCP servers. The LLM agent is implemented using the OpenAI Agent SDK. To guarantee safety, users are asked to approve or reject tool call requests from the agent. Users can enable fully autonomous execution by turning on the auto-approve toggle button, which activates automatic approval of tool calls.

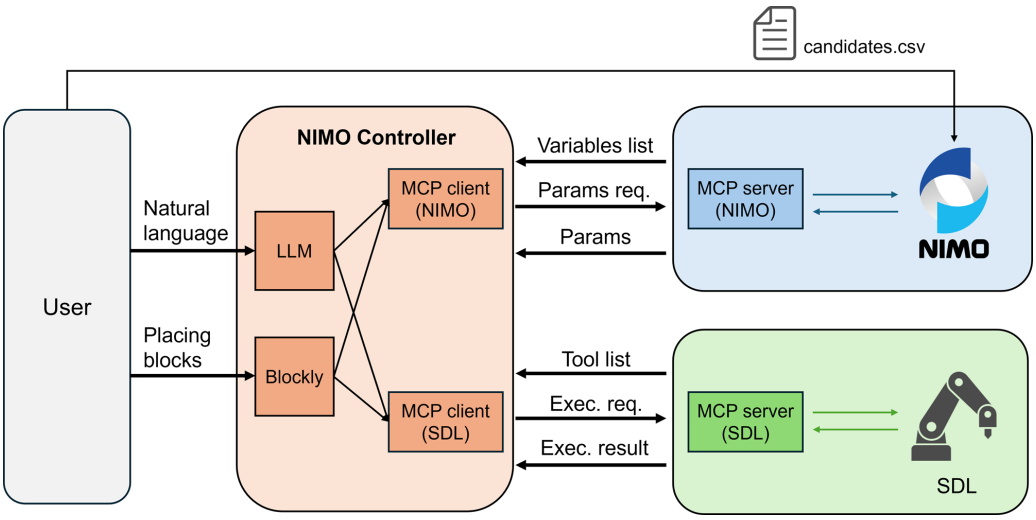

Fig. 1: Overview of the NIMO Controller.

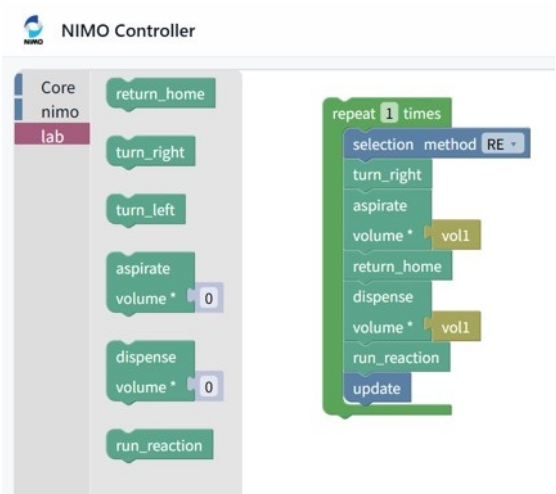

Fig. 2: Screenshot of the visual programming interface.

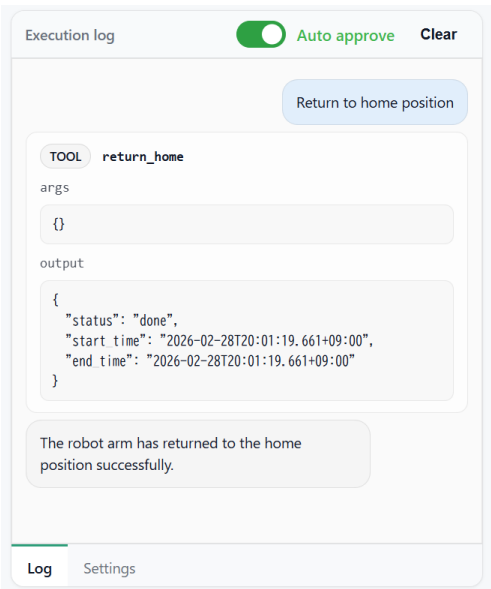

Fig. 3: Screenshot of the natural language interface.

2.3 MCP servers

2.3.1 NIMO MCP server

The NIMO MCP server wraps the NIMO library, which implements decision-making algorithms for materials exploration. The user initially uploads a CSV file (`candidate.csv`) that lists the candidate experimental conditions. The MCP server exposes the parameter names and their currently selected values. It also exposes the `nimo.selection()` function to propose the next experimental conditions and the `nimo.update()` function to update the latest experimental results.

2.3.2 SDL MCP server

The SDL MCP server wraps the functionality of the SDL. This design achieves loose coupling between the frontend and backend interfaces: the implementation of the MCP server is left to the developers, and the SDL can be developed independently without considering the frontend. It enables the integration of legacy systems that do not support Python, as long as the SDL communicates via MCP.

3. Conclusion and future perspective

We have presented NIMO Controller, an MCP-based SDL controller that provides an accessible interface to algorithmic decision-making and automated experimentation through both visual programming and natural language interaction.

Future work includes automatic block-based workflow design through the natural language interface, supporting dynamic updates to MCP servers, implementing a scheduler for concurrent execution of multiple tools, and conducting usability studies with domain scientists to quantify the accessibility gains of the dual-interface approach.

Acknowledgments

This research is supported by JSPS KAKENHI Grant Number JP25K21333.

References

[1] Gary Tom, Stefan P Schmid, Sterling G Baird, Yang Cao, Kourosh Darvish, Han Hao, Stanley Lo, Sergio Pablo-García, Ella M Rajaonson, Marta Skreta, et al. Self-driving laboratories for chemistry and materials science. *Chemical Reviews*, 124(16):9633–9732, 2024.

[2] Wenyu Zhang, Lucy Hao, Veronica Lai, Ryan Corkery, Jacob Jessiman, Jiayu Zhang, Junliang Liu, Yusuke Sato, Maria Politi, Matthew E Reish, et al. IvoryOS: an interoperable web interface for orchestrating Python-based self-driving laboratories. *Nature Communications*, 16(1):5182, 2025.

[3] 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. *Science and Technology of Advanced Materials: Methods*, 3(1):2232297, 2023.

[4] Naruki Yoshikawa, Taiga Ozawa, Wenyu Zhang, Jason Hein, Ryo Tamura, and Matsuda Shoichi. Bridging decision-making engines and workflow design in self-driving laboratories: A NIMO–IvoryOS integration study. *ChemRxiv*, 2026.

[5] Ryo Tamura, Hiromichi Taketa, Satoshi Murata, Daisuke Ryuno, Tomotaka Yokota, Koji Tsuda, and Shoichi Matsuda. Seamless integration of legacy robotic systems into a self-driving laboratory via NIMO: a case study on liquid handler automation. *Science and Technology of Advanced Materials: Methods*, 5(1):2565144, 2025.