Article

https://doi.org/10.1038/s41467-024-54067-7

OCTOPUS: operation control system for task optimization and job parallelization via a user-optimal scheduler

Received: 16 April 2024

Accepted: 29 October 2024

Published online: 08 November 2024

Check for updates

Hyuk Jun Yoo[1,2], Kwan-Young Lee[2] ✉, Donghun Kim[1] ✉ & Sang Soo Han[1] ✉

The material acceleration platform, empowered by robotics and artificial intelligence, is a transformative approach for expediting material discovery processes across diverse domains. However, the development of an operating system for material acceleration platform faces challenges in simultaneously managing diverse experiments from multiple users. Specifically, when it is utilized by multiple users, the overlapping challenges of experimental modules or devices can lead to inefficiencies in both resource utilization and safety hazards. To overcome these challenges, we present an operation control system for material acceleration platform, namely, OCTOPUS, which is an acronym for operation control system for task optimization and job parallelization via a user-optimal scheduler. OCTOPUS streamlines experiment scheduling and optimizes resource utilization through integrating its interface node, master node and module nodes. Leveraging process modularization and a network protocol, OCTOPUS ensures the homogeneity, scalability, safety and versatility of the platform. In addition, OCTOPUS embodies a user-optimal scheduler. Job parallelization and task optimization techniques mitigate delays and safety hazards within realistic operational environments, while the closed-packing schedule algorithm efficiently executes multiple jobs with minimal resource waste. Copilot of OCTOPUS is developed to promote the reusability of OCTOPUS for potential users with their own sets of lab resources, which substantially simplifies the process of code generation and customization through GPT recommendations and client feedback. This work offers a solution to the challenges encountered within the platform accessed by multiple users, and thereby will facilitate its widespread adoption in material development processes.

The material acceleration platform (MAP) has revolutionized material discovery through extensive exploration of the chemical space in diverse domains, including organic molecules, perovskites, colloidal quantum dots, and nanoparticles[1–4]. Through the integration of robotic automation and AI-based experimental planning, MAP has shown promising potential in enhancing the search efficiency for the target materials[5–8] while ensuring reliability in surveillance-free environments[9–12]. However, the scalability and accessibility of MAP with multiple users have been impeded by limitations in experimental device sharing and resource allocation, necessitating advanced research on operating system (OS).

The advent of OS for MAP, exemplified by Chemputer and ChemOS, has provided a promising pathway toward integrated management systems[13,14]. Yet, their accessibility remains restricted and is

[1]Computational Science Research Center, Korea Institute of Science and Technology, Seoul, Republic of Korea. [2]Department of Chemical and Biological Engineering, Korea University, Seoul, Republic of Korea. ✉e-mail: kylee@korea.ac.kr; donghun@kist.re.kr; sangsoo@kist.re.kr

primarily constrained by single-researcher execution and limited applicability. Additionally, the inefficiency of device resource allocations due to the lack of experimental device sharing across different applications poses scalability challenges[15–17]. Addressing these constraints requires a paradigm shift toward a multiuser system, which necessitates the development of a central management platform with a master/node relationship capable of managing experimental equipment efficiently[18]. Central to this endeavor, is the modularization of independent experimental processes to facilitate widespread equipment sharing across diverse applications. However, the development of an OS for multiuser systems has posed challenges to standardized methodologies, primarily stemming from equipment heterogeneity, platform delocalization, and safety concerns[19–27]. Thus, there is a pressing need for extensive research to develop an advanced OS for MAP accessed by multiple users.

When implementing central management systems for processing diverse experiments from multiple users, challenges such as module and device overlap are persistent, leading to resource allocation inefficiencies and safety hazards[16,17]. Module overlap constraints, for instance, result in time wastage and operational inefficiencies, while device overlap issues exacerbate safety concerns from collisions of the activity radius. Addressing these challenges requires a comprehensive scheduling approach that considers resource allocations, safety protocols, and operational efficiencies within realistic environments.

In response to these challenges, we introduce OCTOPUS, an acronym for an operation control system for task optimization and job parallelization via a user-optimal scheduler, which is designed to streamline experimental task scheduling, optimize resource utilization, and enhance safety protocols. OCTOPUS comprises three core components− an interface node, master node, and module nodes− that facilitate client request handling and experimental task scheduling. Leveraging process modularization and a network protocol, OCTOPUS ensures homogeneity, scalability, safety, and versatility within a central management platform. OCTOPUS embodies use-optimal schedulers; it integrates job parallelization techniques to mitigate delays by leveraging device standby times, while the task optimization algorithms prevent safety hazards stemming from device collisions or sharing within realistic operational environments. Additionally, the closed-packing schedule (CPS) algorithm enables module resources to be maximally utilized via compact packing. The user-optimal scheduler (US) in OCTOPUS addresses several challenges encountered within MAP accessed by multiple users and will catalyze its widespread adoption for expedited discovery of novel materials.

Results

Terminology definition

The intricate nature of the individual components within the MAP necessitates a standardized terminology to eliminate potential confusion. In response, recent research has advocated for a comprehensive summary of terms and definitions pertinent to MAP[28]. Expanding upon this effort, we present a detailed elucidation of the structural framework underlying MAP, which encompasses four primary components, platforms, modules, tasks, and actions (Supplementary Fig. S1).

The platform serves as the foundational framework that interconnects experimental resources, facilitating automated experiments to cater to the diverse needs of multiple users. The concept of platforms has been increasingly emphasized in recent advancements[29–31]. Within a platform, modules represent the fundamental building blocks, each dedicated to executing a specific experimental process. Examples of modules include “BatchSynthesis”, “FlowSynthesis”, “SprayCoating”, “Filtration” and “UV–Vis”. Each module encapsulates the requisite functionalities necessary for its designated experimental process.

A task delineates the granular steps comprising a specific experimental process within a module. For instance, tasks within the “BatchSynthesis” module include “PrepareContainer”, “AddSolution”, “Stir”, “Heat”, “Mix” and “React”, among others. Actions, on the other hand, denote the discrete operations of the experimental devices required for task execution. Experimental devices such as robotic arms, pipettes, and stirrers execute various actions to accomplish a task within a module. For example, the “PrepareContainer” task involves sequential actions by a robotic arm of (1) opening vial storage, (2) picking vials and (3) placing vials on a stirrer (Supplementary Fig. S1).

Through this structured framework, a MAP harmoniously orchestrates all the components, empowering users to execute desired experiments with high precision and efficiency. This hierarchical terminology definition not only enhances clarity and comprehension within the field but also facilitates seamless communication and collaboration among researchers and practitioners working within the automated experimentation realm.

Architecture of OCTOPUS

The architectural blueprint of OCTOPUS is depicted in Fig. 1, which highlights and draws inspiration from the adaptive nature of the octopus. OCTOPUS, which comprises three integral components, the interface node, master node, and module nodes, orchestrates the seamless execution of the experimental modules within the MAP.

The interface node serves as the nexus, bridging multiple users (or clients) in remote environments to the MAP infrastructure. Next, the master node plays a central role in managing a myriad of tasks and actions, employing a sophisticated suite of six components, a job scheduler, a task generator, a task scheduler, an action translator, an action executor, and a resource manager. Finally, the module nodes of OCTOPUS are meticulously modularized to accommodate individual experimental processes, as illustrated in Fig. 1. Upon receiving experimental action directives from the master node, the module nodes initiate modularized robotic operations, encompassing a spectrum of actions such as robotic arm maneuvers, solution dispensation and stirrer activation, among others. The hierarchical structure of OCTOPUS (interface node → master node → module nodes) and the harmonious integration of these components elucidate the intricate workflow of the platform.

Job submission via the interface node and job scheduler of the master node

The interface node within OCTOPUS functions as a command line interface (CLI), enabling simultaneous job submissions from multiple clients, as illustrated in Fig. 2. Notably, recent advancements in the user interface design for MAP have primarily been tailored to receiving a single job within an OS[13,14,19–24]. However, the ability to accommodate multiple job submissions concurrently is paramount in enhancing the efficiency of a MAP[25]. Thus, the interface node, bolstered by a multithread pool, enables numerous clients to seamlessly request remote experiments across multiple applications, akin to a computing server (Fig. 2). Notably, OCTOPUS operates real chemical experiments rather than computational tasks, distinguishing it from traditional computing server systems[32,33].

Clients are required to generate job scripts based on the JavaScript object notation (JSON) format, and examples of job scripts are provided in Supplementary Fig. S2. These job scripts cater to a spectrum of client demands, ranging from manual experiments with predefined input conditions (e.g. a model with the name of “Manual” in Supplementary Fig. S2a) to autonomous experiments enabled by AI-decision processes (e.g., model with the names of “BayesianOptimization”, “DecisionTree” and “BayesianNeuralNetwork” in Fig. S2b). The model names could be changed by the clients depending on the type of AI model built for experimental planning on the platform.

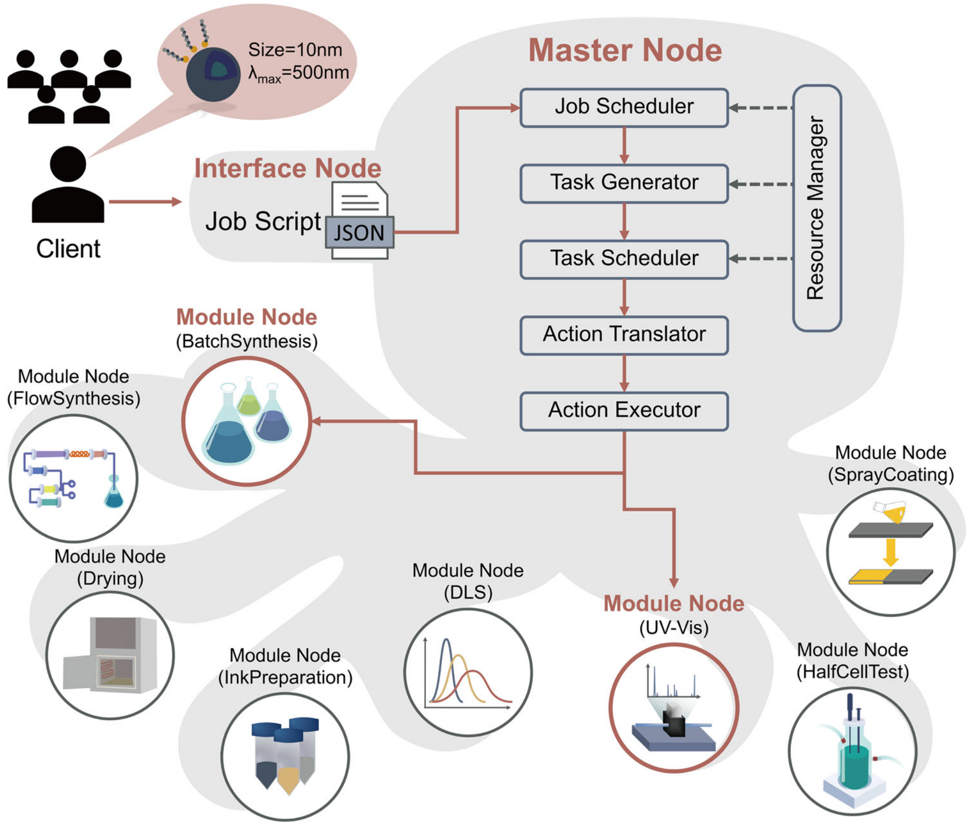


Fig. 1 | Schematic design of OCTOPUS. The architecture of OCTOPUS comprises three main components, the interface node, master node, and module nodes. Multiple clients submit job scripts via CLI based on the JavaScript object notation (JSON) format. The master node manages the submitted jobs, overseeing the job scheduler, task generator, task scheduler, action translator, and action executor. The job scheduler prioritizes and schedules jobs, while the task generator creates tasks based on the received script. The task scheduler provides efficient resource allocations, and the action translator converts task information into device commands. Module nodes operate experimental devices based on instructions from the action executor.

The interface node is responsible for managing the client login process and enforcing stringent security policies through Auth0 API (application programming interface) functions while awaiting client commands (Supplementary Fig. S3). Once authenticated, clients submit job scripts via CLI, leveraging portable batch system commands, with detailed examples provided in Supplementary Fig. S4[32,33]. This transformation of the local platform into a client/server based ubiquitous platform has profound implications for research continuity, particularly in scenarios constrained by temporal and spatial limitations, such as those imposed by COVID-19[34,35].

The master node of OCTOPUS embodies a central management system capable of orchestrating diverse experiments for multiple clients[18]. Within the master node, the job scheduler, which comprises a job ID generator, job modeling unit, job storage, and three distinct queues (waiting, executing, and holding), plays a pivotal role in scheduling jobs for the execution of real experiments, as illustrated in Fig. 2. Upon job submission, the job ID generator assigns a unique identifier (ID) to each job, facilitating orderly processing based on submission order. This job ID is subsequently transferred to the job modeling unit, which generates specific job configurations based on the provided job script, comprising process recommendations (manual vs. autonomous), module and task sequences, and experimental conditions. Subsequently, the job storage organizes the generated jobs, awaiting the triggering decision.

The triggering decision, determined by various factors, including module resource availability and experimental device status, directs the job execution sequence. If the job trigger decides to execute the next job, it transitions the job ID from the waiting queue to the executing queue for real experiment executions (Supplementary Fig. S5). Moreover, a holding queue addresses the safety concerns inherent in surveillance-free MAP environments. In the event of severe safety hazards involving, for example, inflammable or hazardous chemicals[12], the platform automatically places the job on hold, ensuring user safety. Clients retain the autonomy to restart or terminate holding jobs via predefined commands. Several examples of job management (submission, status check, hold, restart, and deletion) via CLI are provided in Supplementary Fig. S6. We also provide Supplementary Movie 1, where the process of logging in, and performing remote and simultaneous job submissions by two users in the interface node of OCTOPUS is recorded for clarity.

Job executions in the master node

Following the job submissions in the interface node, we present the job execution workflow in the master node, incorporating the seven components of job scheduler, task generator, task scheduler, action translator, action executor, resource manager, and database, which are specifically tailored for closed-loop experiments, as illustrated in Fig. 3.

We empower the task generator to retrieve experimental device information, including physical specifications and the setup environment, from the resource manager. For instance, when preparing to execute the "AddSolution" task, the task generator accesses information detailing the device specifications and setup environments of stock solution types and concentrations, among others. Notably, the dynamic updating of experimental device information from the resource manager enhances operational flexibility (Supplementary Fig. S7a and b). The task generator integrates the job script with actual experimental device information to complete the predefined task template (in JSON format) containing the experimental conditions and

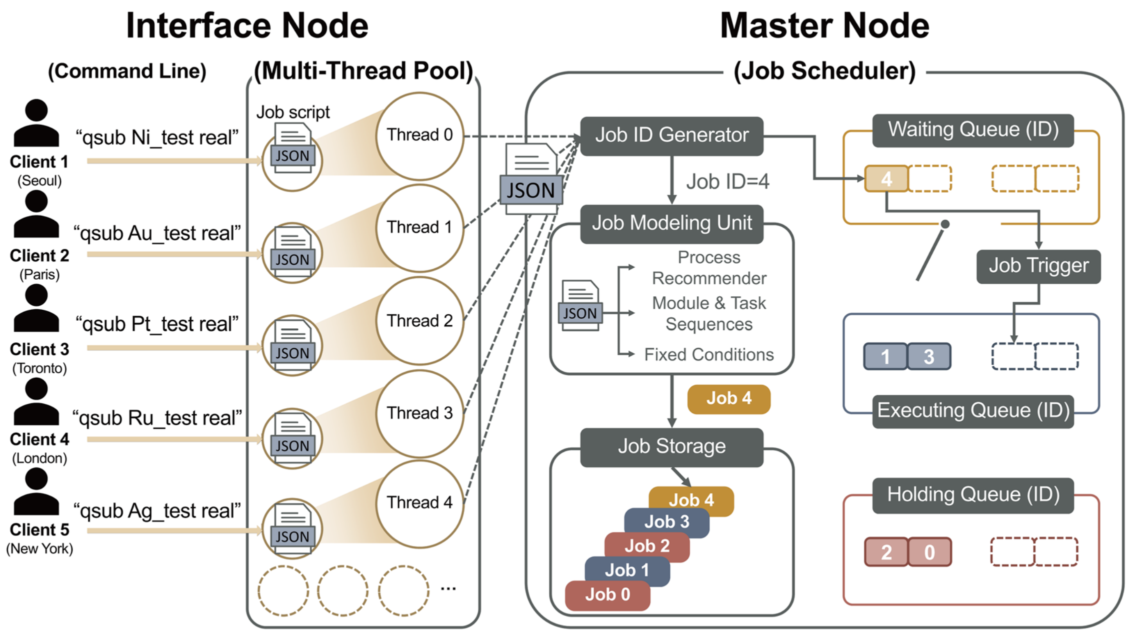


Fig. 2 | Scheme of job submission via the interface node and job scheduler of the master node. The interface node enables clients to submit job scripts, with concurrent handling supported by a multithread pool. The job scheduler in the master node consists of six components, a job ID generator, job modeling unit, job storage, and three queues (waiting, executing, and holding), along with a job trigger. Upon receiving a job script, the job ID generator assigns a unique ID and sends it to the waiting queue and job modeling unit. The job trigger monitors job executions and moves jobs from the waiting queue to the executing queue. Clients can manage the submitted jobs remotely via command lines in the interface node.

task sequences (Fig. 3 and Supplementary Figs. S7c–e). Subsequently, the task scheduler supervises resource allocations for task executions by obtaining information on the location indices from the resource manager (Fig. 3 and Supplementary Fig. S8).

Crucially, the abstraction of tasks, as exemplified in Chemputer[14] and HELAO-async[24], ensures adaptability across different platforms by omitting specific device operation and location details[14,24]. To concretize abstracted tasks, we implement an action translator, which translates abstracted tasks into concrete actions, incorporating predefined action sequences and device location information. For instance, the "AddSolution" task involves the serial actions of initializing a pump, moving a dispenser, and injecting a solution, each executed by different devices (Fig. 3 and Supplementary Fig. S9). Next, the action executor transmits device commands for real experiment execution, employing a transmission protocol that follows predefined data types to module nodes. These data types, including the job ID, device name, action type, action data, and mode type, are encoded and transmitted to module nodes via the Transmission Control Protocol/Internet Protocol (TCP/IP) (Fig. 3 and Supplementary Fig. S10). The modules execute experimental action and return the resulting data to the database for subsequent experimental planning. More detailed information on the resulting data structure is provided in Supplementary Fig. S11.

Network protocol-based modularization

To manipulate multiple jobs within OCTOPUS, process modularization is key for efficiently operating experimental processes. The network protocol was implemented for modularization, which has also been demonstrated in the work of Stein and coworkers[36], for facilitating efficient communication in a brokering system named FINALES (fast intention-agnostic learning server). We conceptualized four key concepts to explain the benefits of using experimental process modularization based on network protocols, which include homogeneity, scalability, safety, and versatility, as illustrated in Fig. 4.

First, a challenge may arise when different manufacturers utilize various programming languages and operating systems for their own experimental devices, and such heterogeneity may hinder seamless communication between module nodes and devices. Examples of the heterogeneous environments among devices in the "BatchSynthesis" and "UV–Vis" modules are provided in Supplementary Fig. S12a. To achieve a homogeneous environment in OCTOPUS, we opted to connect the experimental devices via the TCP/IP technique (Fig. 4a). Since most programming languages, including C, C++, Python, JAVA and JavaScript, support TCP/IP regardless of the operating system (both Linux and Windows), we independently created device servers to facilitate communications between module nodes and devices via TCP/IP, ensuring platform homogeneity (Supplementary Fig. S12b).

Second, the TCP/IP network protocol not only ensures homogeneity but also offers significant advantages in terms of scalability. Scalability in MAP refers to the ability to connect experimental modules of multiple laboratories, even across different nations, as illustrated in Fig. 4b. When multiple experimental modules are joined in a TCP/IP-based network, it logically becomes part of the same network, overcoming spatial constraints in MAP and providing infinite possibilities for platform scalability. Notably, synthesized materials cannot be transferred across different regions due to practical issues, including time costs and material degradation. Thus, the absence of material synthesis modules in a specific laboratory could hinder flexible utilization for diverse applications, despite all experimental modules (modules for material synthesis, characterization, and property evaluation) being connected within a single network. To address this, we proposed customizing material synthesis modules in each laboratory and integrating all the other local modules into a single platform despite remote distances via the TCP/IP-based network protocol (Fig. 4b)[20,21,37,38]. The method for integrating network protocols is detailed in Supplementary Fig. S13.

Third, to minimize safety concerns arising from physical disconnections, we implemented the user datagram protocol (UDP) with TCP/IP to monitor physical connection issues for platform safety, as illustrated in Fig. 4c. If miscommunication occurs between the master node, module nodes due to a physical disconnection between module nodes and experimental devices, the UDP-based broadcast executes

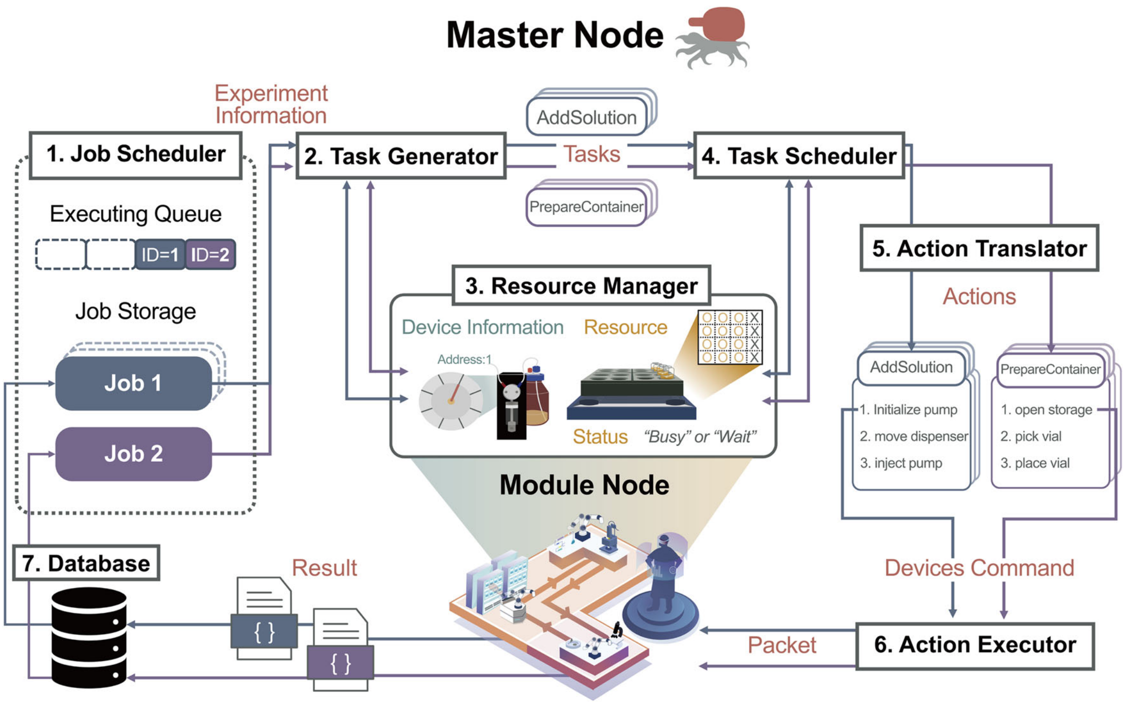


Fig. 3 | Job executions at the master node. Job executions follow a closed-loop experimentation process involving several components, the job in the executing queue, the resource manager, the task generator, the task scheduler, the action translator, the action executor, and the database. When a job is ready for execution, it is matched in the job storage, initiating the experiment. Experiment information is transferred to the task generator, which generates tasks based on experimental conditions and device information. The task scheduler manages task executions, and the action translator converts task information into device commands. After executing the commands, the results are stored in the database, and the job recommends the next experimental conditions in a closed-loop manner.

an emergency stop for the other modules in the internet network to prevent safety issues in surveillance-free systems (Fig. 4c and Supplementary Fig. S14a). In addition to the emergency stop, an alert system based on messengers was implemented to provide researchers with notification and monitoring systems, which aids in swiftly recovering the platform from safety issues (Fig. 4c and Supplementary Fig. S14b–e).

Fourth, a key challenge in MAP is that most platforms have thus far been developed only for a specific purpose; experimental devices are limited to a specific application[14,39–41]. However, with TCP/IP-based process modularization, a static platform can evolve into a versatile platform (Fig. 4d). Researchers can selectively adopt modules tailored to their own applications, allowing customized process design within a single platform and ensuring the versatility of the platform.

Job parallelization to address the module overlap challenge

The compositions of the master node and module nodes and their functionalities are investigated with detailed examples. In the following sections, the user-optimal scheduler (US) developed within OCTOPUS is explained, incorporating three techniques, job parallelization, task optimization, and CPS.

When multiple clients attempt to utilize a MAP with many modules, module overlap issues may occur. Customized scheduling algorithms are necessary to address module overlap issues effectively. One conventional approach is job serialization based on the FCFS (First-Come-First-Serve) concept (Fig. 5a), where the priorities of module usage are assigned based on the job submission order. However, in job serialization, device standby times, where no significant actions are performed in terms of devices, may substantially impair job execution efficiency. An example of device standby times is the chemical reaction period ("React" task) in the "BatchSynthesis" module, where chemical reactions occur in chemical vessels but no actions are performed in terms of devices.

To improve module resource utilization, we introduced job parallelization by leveraging device standby times (Fig. 5a and Supplementary Fig. S15). During the device standby times of a preceding job, the following job can run in parallel without hindering previously started tasks. As illustrated in Fig. 5b, while job ID 0 progresses during the "React" task in the "BatchSynthesis" module, the "PrepareContainer" task in the next job (job ID 1) can be executed simultaneously by leveraging the device standby time (chemical reaction period) of the prior "React" task (Fig. 5b). This specific example of job parallelization is provided in Supplementary Movie 2.

To evaluate the impact of job parallelization on time efficiency, we conducted virtual experiments on catalysis processes, which typically involve long device standby times for chemical reactions. In this test, we define 10 virtual modules related to catalyst synthesis ("BatchSynthesis", "BallMilling"), preprocessing ("Washing", "Filtration", "Drying", "InkPreparation", "SprayCoating"), characterization ("XRD"), and property evaluation ("HalfCellTest", "FullCellTest"). Each module involves both a device execution time and a device standby time, as described in Supplementary Figs. S16 and S17. Seven different types of catalysis experiments utilizing parts of these modules were performed (Fig. 5c, Supplementary Figs. S16 and S17). Figure 5d compares the results between job serialization and parallelization using bar charts. Three performance metrics were implemented, including "job waiting time", "job turnaround time" and "job total time" (Supplementary Fig. S18). Job waiting time reflects the difference between the job submission and start time, providing client-centric performance insights[42]. "Job turnaround time" indicates the duration between the start of a job and completion of a job, and it offers administrator-centric performance insights. "Job total time", the sum of "job waiting time" and "job turnaround time", reflects the overall job execution efficiency.

Job serialization entails the sequential execution of modules based on a priority order. Analysis of the performance metrics for time efficiency reveals that, in serialized jobs, the lower-priority jobs,

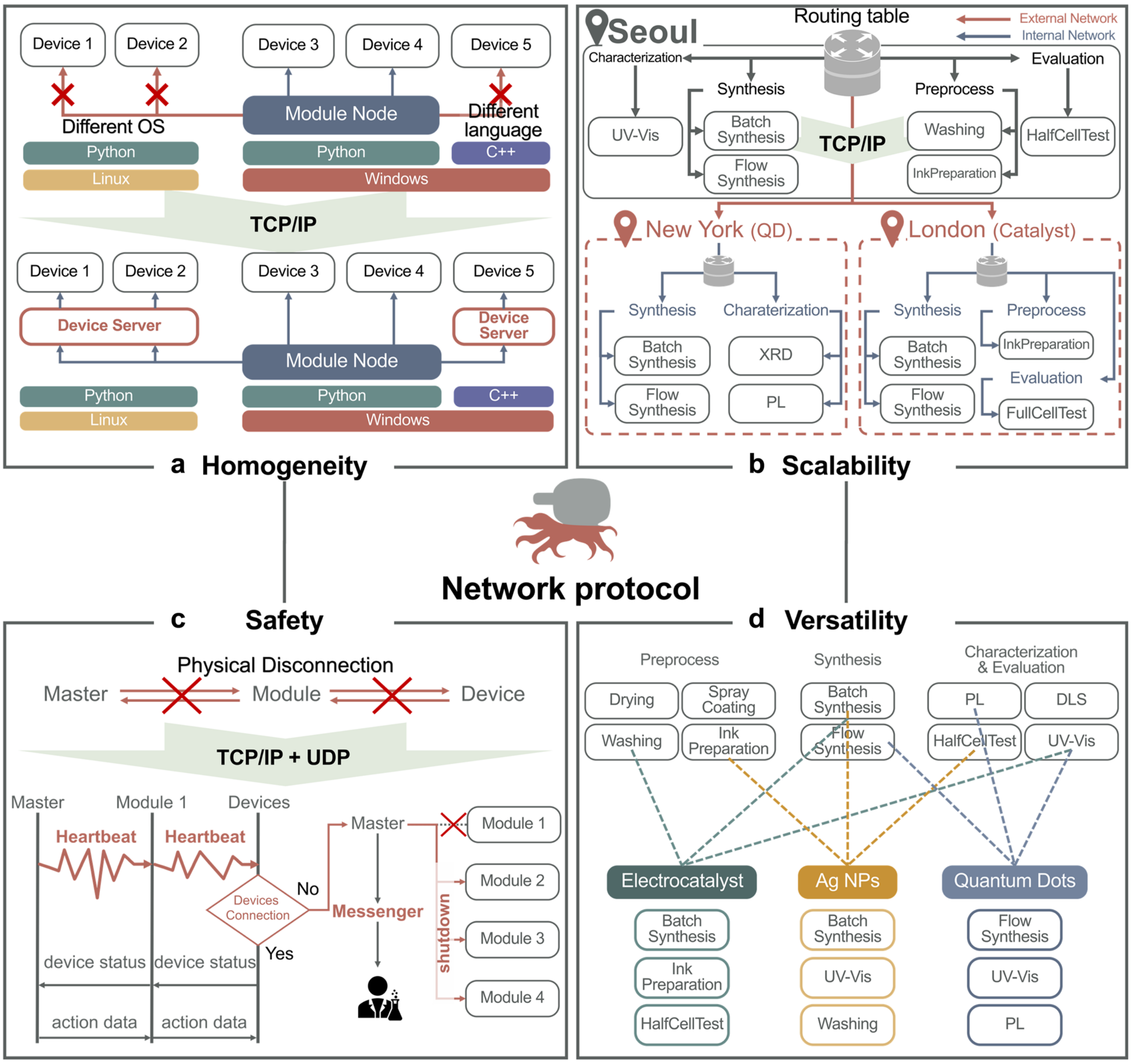


Fig. 4 | Network protocol-aided process modularization. a This figure illustrates the use of a device server based on TCP/IP to create a homogeneous experimental environment, even with multiple devices based on different OS or programming languages. **b** This figure illustrates an example of experimental processes from multiple laboratories across different nations. Modules for material synthesis, characterization, and property evaluation can be joined into a single platform by leveraging internal (red line) or external (blue line) TCP/IP-based networks. **c** The flow chart depicts algorithms for the heartbeat and shutdown function with a UDP-based broadcast, ensuring stability between the master node, module node, and experimental devices. **d** This figure illustrates an example of a versatile platform, where different sets of modules can be used for diverse applications.

determined by job submission order, may experience substantially increased waiting times due to module overlaps. In contrast, job parallelization aims to reduce waiting times by leveraging the device standby times within each module. Consequently, for all seven jobs in these virtual tests, a significant reduction in “job total time” is achieved via job parallelization compared to job serialization (Fig. 5d). For example, for job ID 3, “job total time” decreased significantly with job parallelization (from 10 h to 6.5 h) by leveraging the device standby time of modules utilized in the preceding jobs, such as the “Batch-Synthesis” module of job ID 1. Similarly, in Job ID 4, “job total time” was nearly halved with job parallelization (10 h to 5.5 h), leveraging device standby times in the module, such as the centrifugation of the “Washing” module, oven usage of the “Drying” module and scanning of the “XRD” module, as shown in Supplementary Fig. S16. As a result, job parallelization significantly improves time efficiency, reducing both “job waiting time” and “job turnaround time”, ultimately shortening “job total time” (Fig. 5d and Supplementary Table S1).

Task optimization with masking table for preventing device overlaps

Up to this point, we operated under the assumption that each module functions independently, with the experimental devices operating without interference. However, as multiple clients submit job requests, the likelihood of multiple tasks within the same module running concurrently increases. In such scenarios, experimental device collisions may occur, raising potential safety concerns (Fig. 6a and Supplementary Fig. S19). Moreover, the MAP cost implications are substantial, as each experimental device typically entails significant expenditures to ensure experimental reliability and precision[15–17]. In this regard, implementing device sharing between different modules has become

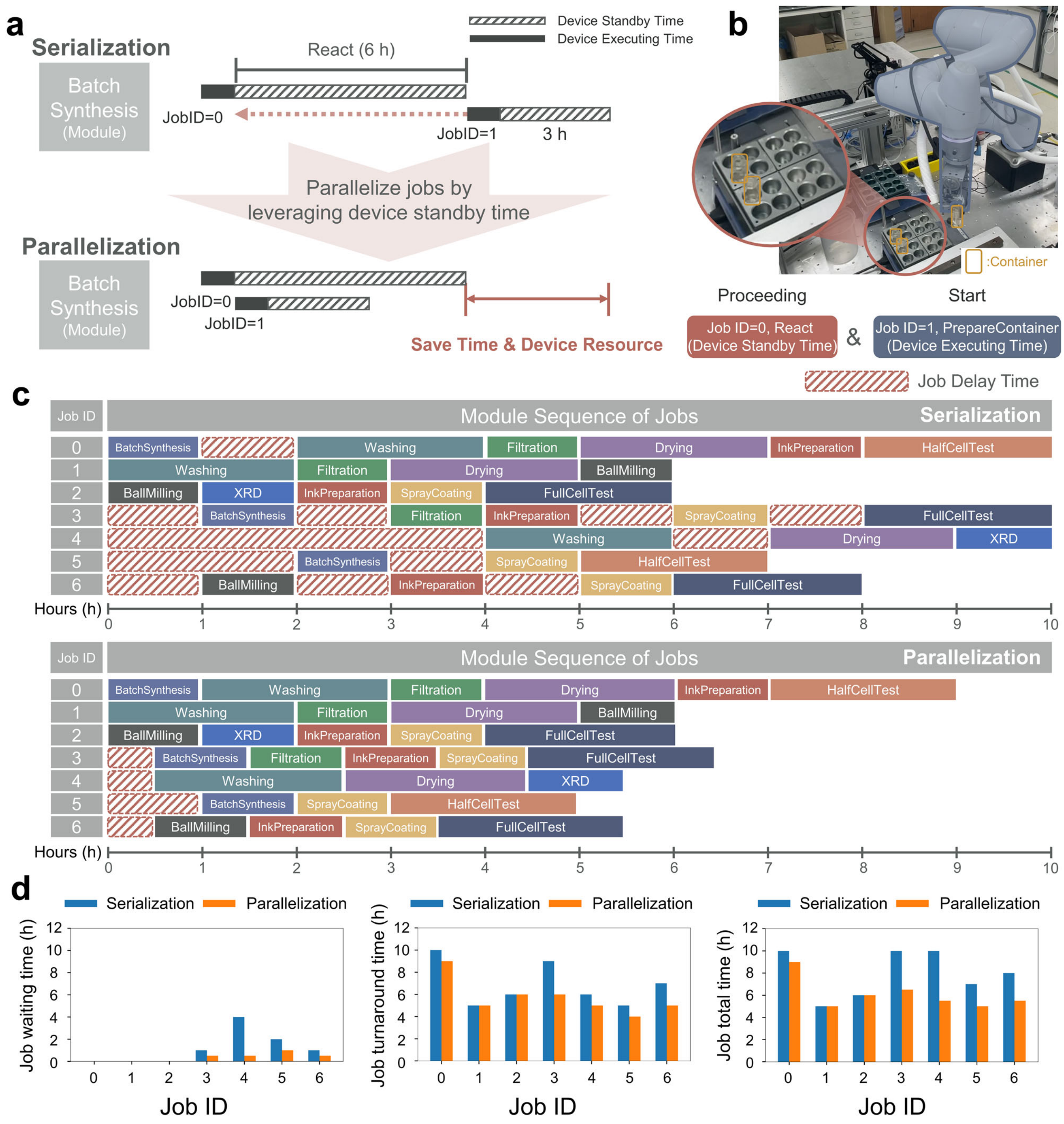
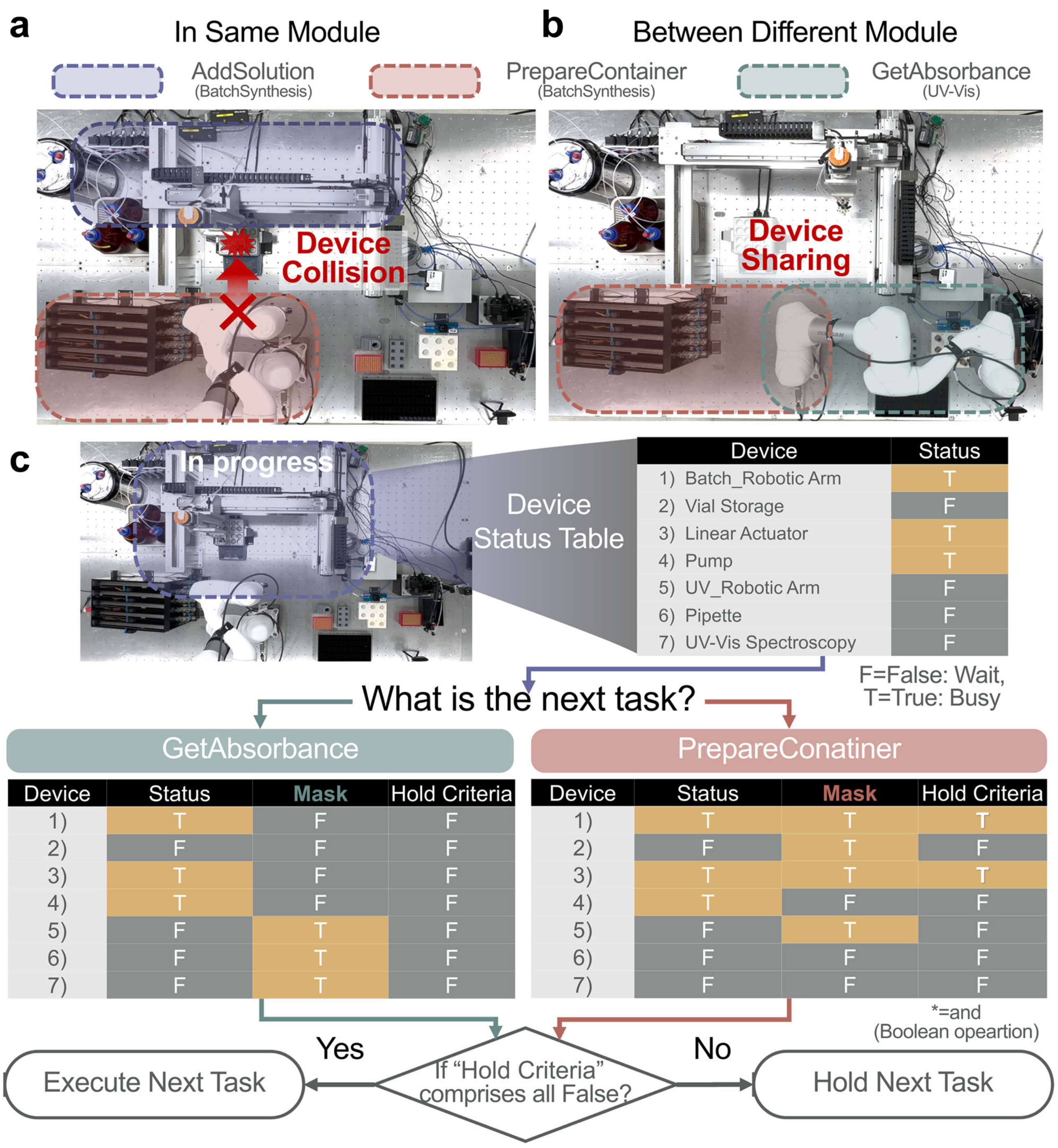


Fig. 5 | Job parallelization to address the module overlap challenge. a Scheme of job serialization and parallelization, showcasing device execution time (black) and device standby time (shaded black). **b** Images representing an example of job parallelization. The enlarged image highlights parallelized batch synthesis. **c** Visualization of module execution results comparing job serialization and parallelization. Shaded red boxes represent the delay time between module executions. **d** Performance metrics of job serialization and parallelization, including the job waiting time, job turnaround time, and job total time. Source data of Fig. 5 are provided in the Source Data file.

imperative for cost savings within limited budgets, as demonstrated in recent advances[30,43–45]. For instance, a robotic arm may serve for both the "BatchSynthesis" and "UV–Vis" modules, as illustrated in Fig. 6b and Supplementary Fig. S19, enabling cost savings through shared utilization[8]. However, when multiple modules attempt to execute individual tasks simultaneously, conflicts may arise regarding the prioritization of tasks for the robotic arm. Consequently, task optimization is urgently needed to prevent device collisions within the same module, and device overlap challenges across different modules.

To overcome these challenges, we introduce a task optimization technique with a masking table. In our approach, each module continuously updates the device status in a tabular format retrieved from the resource manager. Devices currently in use by ongoing tasks are assigned as "True", while unused devices are marked as "False", as shown in the device status table example in Fig. 6c. The resource manager dynamically updates the device status table in real-time. Next, it is essential to predefine masking tables for each task, which identify the devices required for a specific task as "True" and those not needed as "False". For instance, when performing the "GetAbsorbance"



Device	Status
1) Batch_Robotic Arm	T
2) Vial Storage	F
3) Linear Actuator	T
4) Pump	T
5) UV_Robotic Arm	F
6) Pipette	F
7) UV-Vis Spectroscopy	F

GetAbsorbance

Device	Status	Mask	Hold Criteria
1)	T	F	F
2)	F	F	F
3)	T	F	F
4)	T	F	F
5)	F	T	F
6)	F	T	F
7)	F	T	F

PrepareConatiner

Device	Status	Mask	Hold Criteria
1)	T	T	T
2)	F	T	F
3)	T	T	T
4)	T	F	F
5)	F	T	F
6)	F	F	F
7)	F	F	F

Fig. 6 | Task optimization using a masking table. a Example of device overlap challenge - device collision in the same module. **b** Example of device overlap challenge - device sharing between different modules. **c** Schematic algorithm of task optimization with a masking table, employing the AND Boolean operation between the device status table and masking table for a specific task.

task in the "UV–Vis" module, devices such as UV–Vis spectroscopy, robotic arms, and pipettes are involved and are thus marked as "True", while the other devices are marked as "False" to create task-specific masking tables (Fig. 6c). Prior to performing a specific task, the AND Boolean operation is performed between the device status table and the masking table for the task, resulting in a table for "hold criteria" to determine whether to proceed with the next task (Supplementary Fig. S20). If all hold criteria indicate "False", the tasks are executed in parallel, whereas if any of the logical results are found "True", the system waits until the ongoing task is completed, and all the hold criteria indicate "False". Examples of masking tables are provided in Supplementary Fig. S21, and an actual example of a task optimization process with a masking table is provided in Supplementary Movie S3. In summary, when experimental devices are shared between modules, task optimization with masking tables enables cost-saving benefits and enhances the efficiency of device utilization while avoiding device collisions and ensuring safety.

The closed-packing schedule for optimizing module resources

In computing server systems with a finite number of cores, resource allocation considerations are paramount, and are often documented in job scripts[32,33]. Similarly, in the context of MAP, resource management

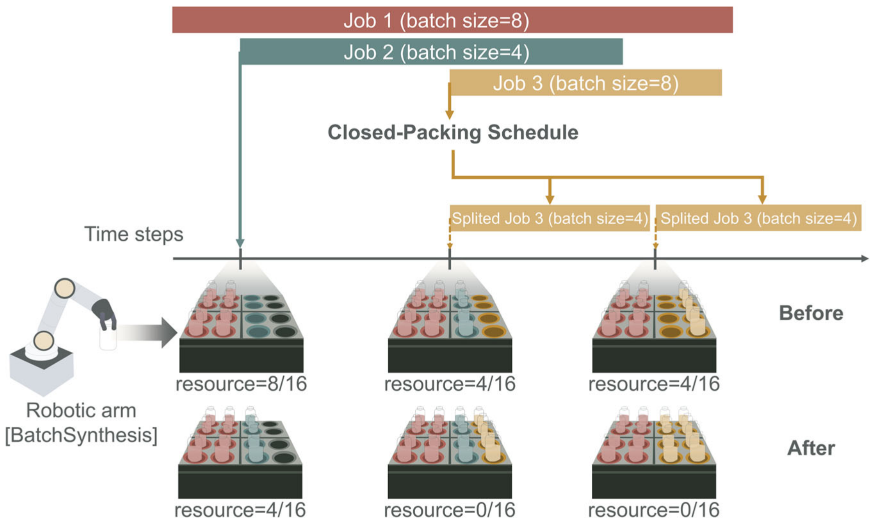


Fig. 7 | Scheme of the closed-packing schedule. The task scheduler with the CPS method aims to utilize the remaining resources of each module efficiently. Job 3, with a batch size of eight, is split into four experiments based on the remaining four stirrer resources in the "BatchSynthesis" module.

extends to modules and experimental devices. Here, device resources denote the maximum number of experiments that each module can accommodate. For example, the "BatchSynthesis" module's resource could be defined by the maximum number of vials simultaneously processable by a magnetic stirrer, while in the "UV–Vis" module, it could also be determined by the maximum number of vial holders available for UV–Vis spectrum measurements (Supplementary Fig. S22). Thus, given the limited availability of resources in each module, scheduling methods that account for these constraints are essential in practical platforms.

To efficiently execute multiple jobs within such constraints, we introduce the closed-packing schedule (CPS) algorithm (Fig. 7 and Supplementary Fig. S23). The core concept of CPS lies in splitting tasks in a single job into multiple batches to maximally utilize the remaining resources via compact packing. An example of CPS is illustrated in Fig. 7, where three jobs (job IDs 1, 2, and 3) are sequentially submitted with different batch sizes (the number of required vials) of eight, four, and eight, respectively, and a stirrer can accommodate up to 16 vials. With active job parallelization, at the execution of job ID 3, 12 resources out of a total of 16 are already occupied by preceding jobs, leaving only 4 resources unused. Then, the CPS divides the tasks of job ID 3 into two batches to allow the first batch to be executed first by fully occupying the 4 unused resources (compact packing), and to allow the other batch to be executed later as soon as the preceding jobs are completed and resources become available. As demonstrated in the example, the CPS aims to effectively minimize the waste of module resources by splitting the tasks within a job based on the computation of the remaining resources.

Performance test of the user-optimal scheduler

In the previous sections, the user-optimal scheduler developed within OCTOPUS was explained and involved three distinct scheduling methods, job parallelization, task optimization, and CPS. To maximize scheduling efficiency, these three scheduling methods were integrated within OCTOPUS, resulting in a united scheduling system named the user-optimal scheduler (US) in this paper. To benchmark the US, we introduced the conventional FCFS to prioritize jobs based on their order of submission (Supplementary Fig. S24)[46]. FCFS executes jobs sequentially based on a priority order. We chose the FCFS algorithm as the baseline scheduler for comparison due to its fairness in executing jobs based on its priority order, which is crucial in multiclient systems.

Figure 8a, b visualizes the execution time efficiency in processing multiple jobs across the two different scheduling schemes of FCFS and US. We employed a platform capable of simulating device collision and sharing issues to reflect a realistic environment[8]. In these comparative tests, 11 jobs with different batch sizes are submitted sequentially, all of which require the use of either the "BatchSynthesis" or "UV–Vis" module (Supplementary Fig. S25). It is important to clearly understand that, in the tests, the techniques of job parallelization, task optimization, and CPS-based resource allocation, developed before, are active only for the US, whereas they are not active in the FCFS scheduling scheme.

Through the observation of the scheduling results, we demonstrate the efficacy of the US, particularly in terms of "job waiting time". The benefits of combined job parallelization and CPS are pronounced in many cases. For example, job IDs 1, 8, and 10 significantly reduced "job waiting time" (8.92 h to 0.63 h for job ID 1, 10.21 h to 0.72 h for job ID 8, and 10.97 h to 0 h for job ID 10). The time reductions are attributed to both leveraging device standby times and splitting jobs via CPS. Notably, these jobs are parallelized during the "React" task (device standby time) in the "BatchSynthesis" module of the preceding jobs (Fig. 8b, Supplementary Fig. S26 and Table S2). Similarly, job ID 4 also benefits from both job parallelization and CPS, leading to a reduction in "job waiting time", as shown in Fig. 8c (11.72 h to 0 h for job ID 4). This job is parallelized with the preceding job ID 3 as the CPS splits the job according to the remaining resources of the "UV–Vis" modules (Fig. 8b and Supplementary Fig. S27). Meanwhile, the benefit of the task optimization technique is well observed in many cases, such as job IDs 2, 3, 4, 6, 7, and 9. Since the "BatchSynthesis" module and the "UV–Vis" module share a robotic arm, the simultaneous execution of these two modules may cause safety hazards from the device sharing environments. However, with active task optimization, concurrent executions of both modules were safely performed, as demonstrated in Supplementary Fig. S28, Table S2, and Movie S3.

By minimizing "job waiting time", the US also proved more efficient in terms of "job total time" compared to the conventional FCFS scheme. However, we observed an overall and slight increase in "job turnaround time" for the US, potentially resulting from a duplication of device standby time (Supplementary Fig. S29). For Job IDs 1, 8, and 10, our scheduling system duplicates the device standby time associated with the "React" task, as the jobs are split based on the remaining resources. This redundancy contributes to the accumulation of "job turnaround time". Overall, while the US significantly reduces "job waiting time", some delays in "job turnaround time" may occur due to device standby time duplications.

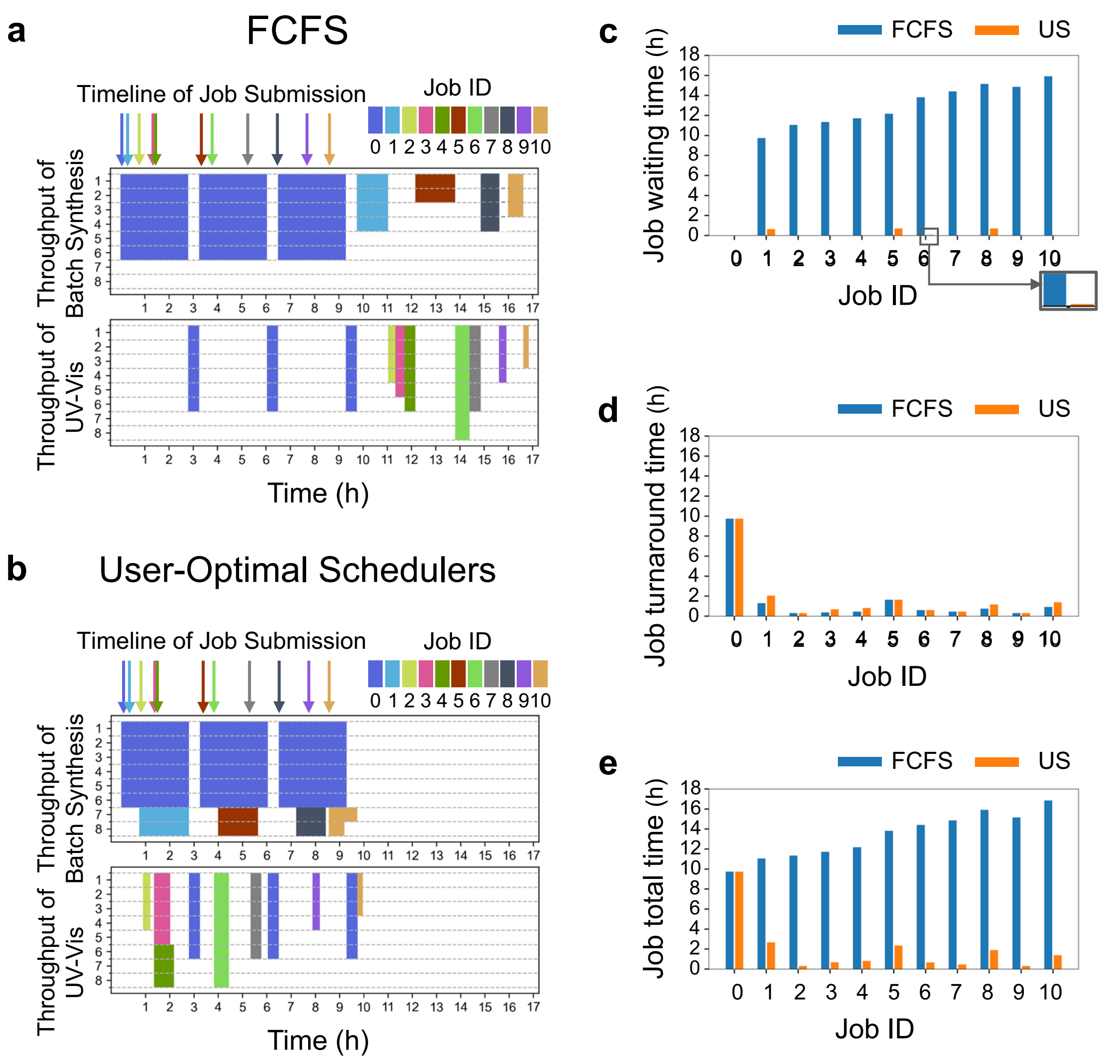


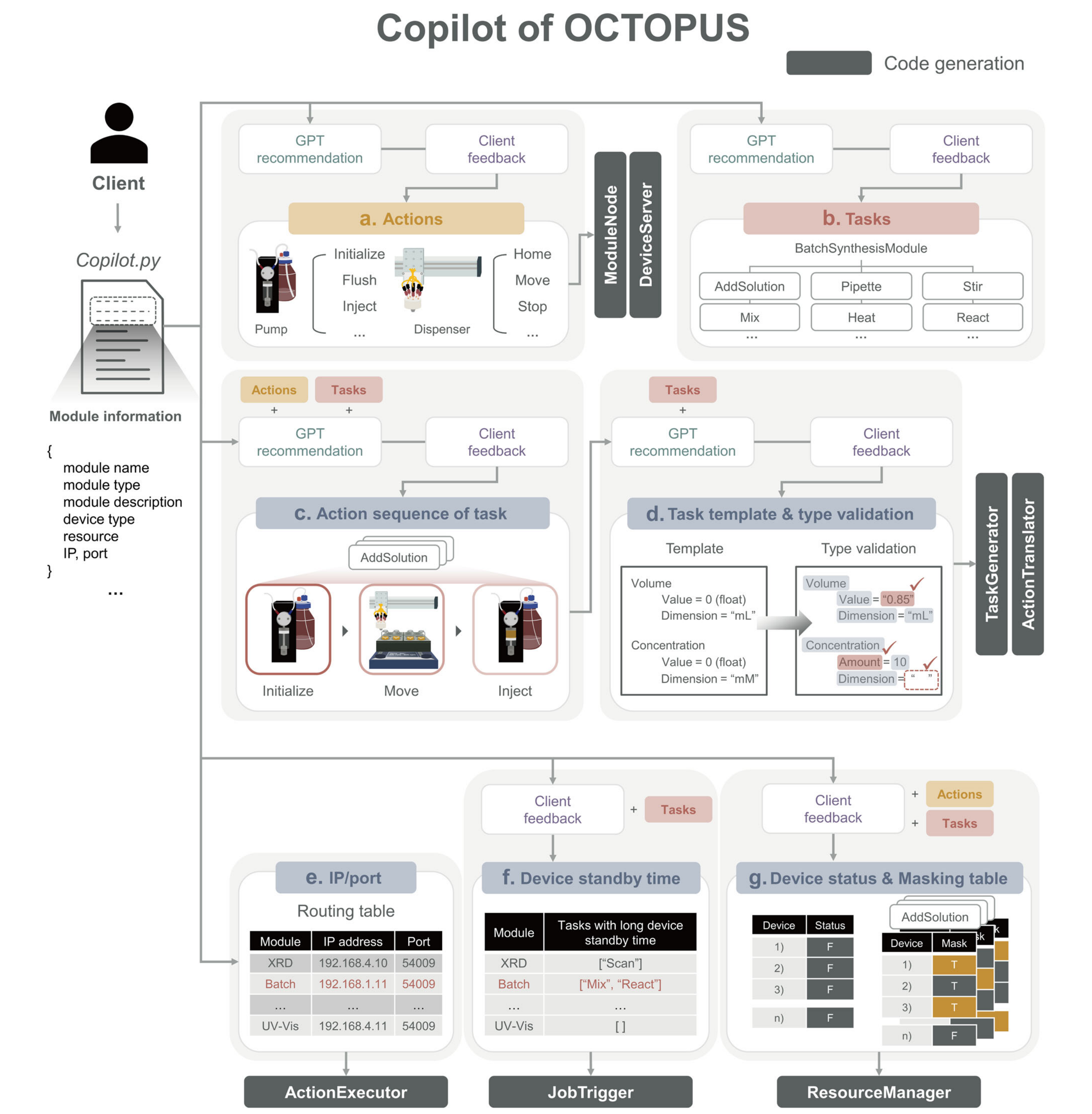
Fig. 8 | Performance comparison of scheduling algorithms for multiple jobs. **a**, **b** Job execution timelines for 11 jobs for first-come-first-served (FCFS) and user-optimal scheduler (US). **c**–**e** Performance metrics of "job waiting time", "job turnaround time" and "job total time" for FCFS and US. Source data of Fig. 8 are provided in the Source Data file.

Overall, by analyzing the performance metrics for time efficiency, the US emerged as highly effective, saving time across all jobs in terms of "job waiting time" (Fig. 8c). Although there is a slight increase in "job turnaround time" due to duplications of device standby time by CPS (Fig. 8d and Supplementary Fig. S27), the significant improvement in efficiency in terms of "job waiting time" renders the US more efficient, leading to a substantial reduction in "job total time", compared to conventional FCFS (Fig. 8e and Supplementary Table S2).

Copilot of OCTOPUS

If a client wants to use the OCTOPUS system for a new set of lab resources, it will require a lot of hands-on code generation and customization for the integration within OCTOPUS. A partial or complete modification is required for generating components of the task generator, action translator, action executor, resource manager, and module node, as illustrated in Supplementary Fig. S30. To improve the reusability of OCTOPUS, we developed "Copilot of OCTOPUS" to offer a convenient module registration process via automated code generations (Fig. 9). Copilot of OCTOPUS features GPT (generative pre-trained transformer)-based recommendations and client feedback to streamline module generation and validation. In the GPT prompt design, a few-shot learning with several examples was performed to enhance its prediction accuracy, and more details on prompt engineering are described in Supplementary Figs. S31–S34.

Copilot of OCTOPUS auto-generates the required codes and files to run OCTOPUS, and the process is as follows. The first step is that a client needs to input simple module information in the 'copilot.py' Python file. The execution of 'copilot.py' generates a list of actions and tasks based on the input module information through GPT recommendations, and then the client reviews and modifies the results (Fig. 9a, b and Supplementary Figs. S31 and S32), ensuring the adaptation to diverse applications and user preferences. Next, the action sequences for a specific task as well as task templates are generated via GPT recommendations and client feedback (Fig. 9c, d and Supplementary Fig. S33). In this stage, the data type validations are generated to check for invalid data in the task template, using Pydantic (https://github.com/pydantic/pydantic) to predefine the JSON data type (Fig. 9d and Supplementary Fig. S34). Lastly, it updates the routing table for managing IP addresses and ports for different modules, identifies the tasks with long device standby times, and completes the device status table and masking tables with client feedback. As a result of these sequential processes, all required codes and files are auto-generated, as described in Supplementary Table S4. The description of inputs and outputs of GPT was demonstrated in Supplementary Table S3. New users can now easily generate diverse codes for a new set of lab



Module	IP address	Port
XRD	192.168.4.10	54009
Batch	192.168.1.11	54009
...	...	...
UV-Vis	192.168.4.11	54009

Module	Tasks with long device standby time
XRD	["Scan"]
Batch	["Mix", "React"]
...	...
UV-Vis	[ ]

Device	Status
1)	F
2)	F
3)	F
n)	F

Device	Mask
1)	T
2)	T
3)	T
n)	F

Fig. 9 | Copilot of OCTOPUS. Copilot of OCTOPUS consists of seven steps for code generation and customization in OCTOPUS. Gray boxes represent the generated codes. **a** Action generation: Device actions are facilitated through GPT recommendations and a client feedback system. Code generation occurs within the module node and device server. **b** Task generation: Module tasks are generated through GPT recommendations and the client feedback system. **c** Action sequence generation: Action sequences for tasks are generated through GPT recommendations and the client feedback system. Before the GPT modeling process begins, the generated tasks and actions are added to the GPT prompt. **d** Task template and type validation: Task templates and type validations are generated through GPT recommendations and the client feedback system. Before the GPT modeling process begins, the generated tasks are added to the GPT prompt. Code generation takes place in the task generator and action translator. **e** IP address and port number registration: This step establishes connections with the module node. Code generation occurs in the action executor. **f** Task registration with long device standby time: Tasks with long device standby times are registered through the client feedback system using the generated tasks. Code generation takes place in the job trigger. **g** Device registration: New devices are registered in the device status table and masking table for task execution through the client feedback system, using the generated tasks and actions. Code generation takes place in the resource manager.

resources via Copilot of OCTOPUS, requiring only minimal modifications to 'copilot.py'. Supplementary Software is appended to this article, providing a detailed and step-by-step protocol to assist potential clients in using Copilot of OCTOPUS without difficulty.

Copilot of OCTOPUS automates the code generation and customization for new module registration, requiring only simple input regarding module information from clients. Copilot of OCTOPUS streamlines operations across different lab setups by reducing the need for manual adjustments, minimizing human error, and enabling seamless task execution (Supplementary Figs. S30 and S34). This automated process not only enhances operational efficiency for module registration but also ensures the accuracy and reliability of module integration within OCTOPUS. The significantly enhanced reusability of OCTOPUS supports scalable MAP evolution and meets growing industrial demands.

Although Copilot of OCTOPUS simplifies the process of code generation and customization through GPT recommendations and client feedback, the accuracy of these recommendations is significantly influenced by the predefined prompts. Recent advancements[47] in prompt engineering have demonstrated that well-customized initial prompts can enhance the quality of GPT responses. To achieve accurate GPT recommendations, further research into advanced prompt engineering for the Copilot of OCTOPUS is necessary.

Discussion

CLI presents a notable departure from conventional experimentation interfaces, potentially posing challenges for traditional experimental experts. While familiar to researchers in computer science fields, its adoption may present difficulties for those accustomed to hands-on experimentation. Recent studies advocate for the development of a user-friendly web-based interface accessible to both experimental researchers and computational experts[14,19,22,24]. Additionally, an advanced relational database management system for MAP should be developed to accommodate differently-formatted material data using JSON, as specified by various chosen modules.

Moreover, the integration of XR (extended reality), including VR[48] (virtual reality) and AR[49] (augmented reality), holds significant potential to greatly enhance the accessibility and usability of OCTOPUS[15,50–52]. VR is defined as a technology that immerses users in a completely virtual environment, and recent advancements show the potential implementation of VR in MAP. The benefits of VR for MAP include replicating client hand motions via remote control. Thus, VR can be utilized for recovery motion planning in the event of robotic device failures. The implementation of VR helps prevent safety accidents and democratizes client interactions through surveillance-free MAP.

Additionally, AR is defined as a technology that overlays virtual elements onto real-world environments. In MAP, the AP technology with its specialized visualization capabilities, can assist in decision-making by displaying job status, safety alerts, AI-decision processes, and knowledge graphs of accumulated data.

However, XR technologies are highly device-dependent, leading to considerable variation in price, performance, and interfacing methods across different manufacturers. Additionally, there is a notable lack of real-time data processing infrastructure for XR devices. These technical bottlenecks hinder the implementation of XR technologies within MAP today. Therefore, standardized interfacing protocols for integrating heterogeneous XR devices are needed, along with the development of real-time data processing capabilities using 4 G/5 G networks and blockchain-based encrypted communication infrastructure.

In conclusion, OCTOPUS embodies a multifaceted solution that has been engineered to overcome the challenges inherent in MAP accessed by multiple users, OCTOPUS embodies a multifaceted solution. First, its tripartite structure, comprising the interface node, master node, and module nodes, orchestrates seamless client request handling and experimental task scheduling. Through the integration of process modularization and network protocol utilization, OCTOPUS establishes a foundation characterized by homogeneity, scalability, safety, and versatility within a central management platform. Furthermore, OCTOPUS presents the US. The incorporation of job parallelization techniques serves to alleviate delays, while task optimization algorithms prevent safety hazards potentially arising from device collisions and sharing. In addition, the development of the CPS algorithm within OCTOPUS represents a significant stride in efficiently executing multiple jobs with minimal resource wastage. Copilot of OCTOPUS is provided to promote the reusability of OCTOPUS for potential users, which significantly simplifies the process of code generation and customization with GPT recommendations and client feedback. OCTOPUS will facilitate the management of diverse experiments from multiple users and thereby accelerate the widespread adoption of MAP for expedited material development.

Methods

Virtual experiments for job parallelization leveraging device standby times

In the virtual experiments related to Fig. 5, we defined the duration of both the device execution time and device standby time of each module as follows. For the "BatchSynthesis", "Filtration", "BallMilling", "InkPreparation", "XRD" and "SprayCoating" modules, a duration of 0.5 h was assigned to both device execution time and device standby time, resulting in 1 h of total time for each module (Supplementary Fig. S16). For the "Washing" module, which is known for its repetitive removal of impurities, each 0.5-h duration was assigned to one of the device execution times or device standby times, resulting in 2 h of total time (Supplementary Fig. S16). For the "Drying", "HalfCellTest", and "FullCellTest" modules, which are known for their extended durations for bottleneck processes, durations of 0.5 h and 1.5 h, respectively, were assigned to each device execution time and device standby time, resulting in 2 h of total time (Supplementary Fig. S16). The time allocations in each module are provided in detail in Supplementary Fig. S16. The experimental devices within each module were assumed to function without mutual interference. The combinations and sequences of modules assigned to each job were carefully chosen based on actual experimental processes, as illustrated in Supplementary Fig. S17. For example, for job ID 0 in Fig. 5, an experiment of synthesizing and measuring a Cu catalyst for the CO_2 reduction reaction involves the following modules in order: "BatchSynthesis", "Washing", "Filtration", "Drying", "InkPreparation" and "HalfCellTest". The modules used in other jobs are also described in Supplementary Fig. S17. The virtual experiment data of job parallelization in this study are provided in the Supplementary Table S1 and Source Data file.

Experiments for task optimization with masking table

Prior to conducting experiments related to Fig. 6, we predefine the masking tables for each task. For example in the "BatchSynthesis" and "UV–Vis" modules, as illustrated in Supplementary Fig. S19, the masking tables for a task represent the usage of experimental devices during the task, including the robotic arm (shared between two modules), vial storage, linear actuator with solution dispenser, syringe pump, pipette, UV–Vis spectroscopy. For example, the "AddSolution" task in the "BatchSynthesis" module involves the activation of a linear actuator and pump; thus, these two devices are marked as "True" and all the other devices are marked as "False" in the masking table. The masking tables are structured with Boolean values (True or False). Examples of the masking table are provided in Supplementary Fig. S21. In this work, the module and device configuration utilized in this task optimization performance test adhered to our prior publication[8].

Experiments for benchmarking user-optimal scheduler

The module and device configuration employed in the performance benchmarking test in Fig. 8 are consistent with recent advancements documented in our previous work[8]. Scheduling schemes of FCFS and US are compared in realistic experimental environments. All the resource information was stored and periodically updated by a resource manager at each module node by using a location index based on the listed data types (Supplementary Fig. S8). Job parallelization, task optimization, and CPS-based resource allocation are active only for the US, whereas they are not active in the FCFS scheduling schemes. In these benchmark tests, the 11 job scripts are submitted based on job submission timelines, as illustrated in Supplementary Fig. S25 and Supplementary Information/Source Data file. These 11 job scripts contain information on the experiment type (model names of manual vs. AI optimizations), module selection ("BatchSynthesis" or "UV–Vis"), batch size, number of closed-loop cycles and task configuration (number of task executions and device standby time). The performance test data of the user-optimal scheduler in this study are provided in Supplementary Table S2 and the Source Data file.

Data availability

Several examples of our result data, the codes, and related explanations are provided in the following GitHub repository (https://github.com/KIST-CSRC/Octopus) and Zenodo[53]. Source data are provided in the Source Data file and Zenodo[53]. Source data are provided with this paper.

Code availability

Several examples of codes and related explanations are provided in the following GitHub repository (https://github.com/KIST-CSRC/Octopus) and Zenodo[53]. All codes are written in Python 3.7 and all environments can be created via requirements.txt file.

References

1. Higgins, K., Valleti, S. M., Ziatdinov, M., Kalinin, S. V. & Ahmadi, M. Chemical robotics enabled exploration of stability in multi-component lead halide perovskites via machine learning. *ACS Energy Lett.* **5**, 3426–3436 (2020).
2. Epps, R. W. et al. Artificial chemist: an autonomous quantum dot synthesis bot. *Adv. Mater.* **32**, 2001626 (2020).
3. Mekki-Berrada, F. et al. Two-step machine learning enables optimized nanoparticle synthesis. *npj Comput. Mater.* **7**, 55 (2021).
4. Angelone, D. et al. Convergence of multiple synthetic paradigms in a universally programmable chemical synthesis machine. *Nat. Chem.* **13**, 63–69 (2021).
5. Häse, F., Roch, L. M., Kreisbeck, C. & Aspuru-Guzik, A. Phoenics: a Bayesian optimizer for chemistry. *ACS Cent. Sci.* **4**, 1134–1145 (2018).
6. Aldeghi, M., Häse, F., Hickman, R. J., Tamblyn, I. & Aspuru-Guzik, A. Golem: an algorithm for robust experiment and process optimization. *Chem. Sci.* **12**, 14792–14807 (2021).
7. Häse, F., Aldeghi, M., Hickman, R. J., Roch, L. M. & Aspuru-Guzik, A. Gryffin: an algorithm for Bayesian optimization of categorical variables informed by expert knowledge. *Appl. Phys. Rev.* **8**, 031406 (2021).
8. Yoo, H. J. et al. Bespoke metal nanoparticle synthesis at room temperature and discovery of chemical knowledge on nanoparticle growth via autonomous experimentations. *Adv. Funct. Mater.* **34**, 2312561 (2024).
9. Yoshikawa, N., Darvish, K., Vakili, M. G., Garg, A. & Aspuru-Guzik, A. Digital pipette: open hardware for liquid transfer in self-driving laboratories. *Digit. Discov.* **2**, 1745–1751 (2023).
10. Yoshikawa, N. et al. Large language model for chemistry robotics. *Auton. Robots.* **47**, 1057–1086 (2023).
11. Jiang, Y. et al. Autonomous biomimetic solid dispensing using a dual-arm robotic manipulator. *Digit. Discov.* **2**, 1733–1744 (2023).
12. Tiong, L. C. O. et al. Machine vision-based detections of transparent chemical vessels toward the safe automation of material synthesis. *npj Comput. Mater.* **10**, 42 (2024).
13. Sim, M., Ghazi Vakili, M., Hao, H., Hickman, R. J. & Pablo-García, S. ChemOS 2.0: an orchestration architecture for chemical self-driving laboratories. *Matter* **7**, 2959–2977 (2024).
14. Steiner, S. et al. Organic synthesis in a modular robotic system driven by a chemical programming language. *Science* **363**, 6423 (2019).
15. Abolhasani, M. & Kumacheva, E. The rise of self-driving labs in chemical and materials sciences. *Nat. Synth.* **2**, 483–492 (2023).
16. Maffettone, P. M. et al. What is missing in autonomous discovery: open challenges for the community. *Digit. Discov.* **2**, 1644–1659 (2023).
17. Seifrid, M. et al. Autonomous chemical experiments: challenges and perspectives on establishing a self-driving lab. *Acc. Chem. Res.* **55**, 2454–2466 (2022).
18. Sayfan, G. *Mastering Kubernetes*. (Packt Publishing Ltd, 2017).
19. van der Westhuizen, C. J., du Toit, J., Neyt, N., Riley, D. & Panayides, J. L. Use of open-source software platform to develop dashboards for control and automation of flow chemistry equipment. *Digit. Discov.* **1**, 596–604 (2022).
20. Rahmanian, F. et al. Enabling modular autonomous feedback-loops in materials science through hierarchical experimental laboratory automation and orchestration. *Adv. Mater. Interfaces* **9**, 2101987 (2022).
21. Strieth-Kalthoff, F. et al. Delocalized, asynchronous, closed-loop discovery of organic laser emitters. *Science* **384**, 6697 (2024).
22. Hielscher, M. M., Dörr, M., Schneider, J. & Waldvogel, S. R. LABS: Laboratory automation and batch scheduling – a modular open source Python program for the control of automated electrochemical synthesis with a web interface. *Chem. Asian J.* **18**, e202300380 (2023).
23. Tamura, R., Tsuda, K. & Matsuda, S. NIMS-OS: an automation software to implement a closed loop between artificial intelligence and robotic experiments in materials science. *Sci. Tech. Adv. Mat* **3**, 1 (2024).
24. Guevarra, D. et al. Orchestrating nimble experiments across interconnected labs. *Digit. Discov.* **2**, 1806–1812 (2023).
25. Kusne, A. G. & McDannald, A. Scalable multi-agent lab framework for lab optimization. *Matter* **6**, 1880–1893 (2023).
26. Deneault, J. R. et al. Toward autonomous additive manufacturing: Bayesian optimization on a 3D printer. *MRS Bull.* **46**, 566–575 (2021).
27. Campbell, S. I. et al. Outlook for artificial intelligence and machine learning at the NSLS-II. *Mach. Learn. Sci. Technol.* **2**, 1 (2021).
28. Leong, C. J. et al. An object-oriented framework to enable workflow evolution across materials acceleration platforms. *Matter* **5**, 3124–3134 (2022).
29. Du, X. et al. Elucidating the full potential of OPV materials utilizing a high-throughput robot-based platform and machine learning. *Joule* **5**, 495–506 (2021).
30. Coley, C. W. et al. A robotic platform for flow synthesis of organic compounds informed by AI planning. *Science* **365**, 6453 (2019).
31. Jiang, Y. et al. An artificial intelligence enabled chemical synthesis robot for exploration and optimization of nanomaterials. *Sci. Adv.* **8**, 1–12 (2022).
32. Yoo, A. B., Jette, M. A. & Grondona, M. Slurm: Simple Linux utility for resource management. *Workshop Job Sched. Strateg. parallel Process.* **44**, 60 (2003).
33. Nabrzyski, J., Schopf, J. M. & Weglarz, J. *Grid Resource Management: State of the Art and Future Trends*. (Springer Science & Business Media, 2012).

34. Vasel, K. The pandemic forced a massive remote-work experiment. Now comes the hard part. *CNN Business* (2021).
35. Park, J. et al. Closed-loop optimization of nanoparticle synthesis enabled by robotics and machine learning. *Matter* **6**, 677–690 (2023).
36. Vogler, M. et al. Brokering between tenants for an international materials acceleration platform. *Matter* **6**, 2647–2665 (2023).
37. Canty, R. B. & Jensen, K. F. Sharing reproducible synthesis recipes. *Nat. Synth*. **3**, 428–429 (2024).
38. Rauschen, R., Guy, M., Hein, J. E. & Cronin, L. Universal chemical programming language for robotic synthesis repeatability. *Nat. Synth*. **3**, 488–496 (2024).
39. Granda, J. M., Donina, L., Dragone, V., Long, D. L. & Cronin, L. Controlling an organic synthesis robot with machine learning to search for new reactivity. *Nature* **559**, 377–381 (2018).
40. Volk, A. A. et al. AlphaFlow: autonomous discovery and optimization of multi-step chemistry using a self-driven fluidic lab guided by reinforcement learning. *Nat. Commun*. **14**, 1403 (2023).
41. Soldatov, M. A. et al. Self-driving laboratories for development of new functional materials and optimizing known reactions. *Nanomaterials* **11**, 619 (2021).
42. Rubab, S., Hassan, M. F., Mahmood, A. K. & Shah, S. N. M. Adoptability study of bin-packing for scheduling jobs on volunteer grid resources. In *Procedia Computer Science* **69**, 2–12 (Elsevier B.V., 2015).
43. MacLeod, B. P. et al. A self-driving laboratory advances the Pareto front for material properties. *Nat. Commun*. **13**, 995 (2022).
44. Burger, B. et al. A mobile robotic chemist. *Nature* **583**, 237–241 (2020).
45. MacLeod, B. P. et al. Self-driving laboratory for accelerated discovery of thin-film materials. *Sci. Adv*. **6**, 1–8 (2020).
46. Putera, A. & Siahaan, U. Comparison Analysis of CPU Scheduling: FCFS, SJF and Round Robin. *Int. J. Eng. Dev. Res*. **4**, 124–132 (2016).
47. Hoon Yi, G. et al. MaTableGPT: GPT-based table data extractor from materials science literature. Preprint at https://doi.org/10.48550/arXiv.2406.05431 (2024).
48. Skibba, R. Virtual reality comes of age. *Nature* **553**, 402–403 (2018).
49. Matthews, D. Virtual-reality applications give science a new dimension. *Nature* **557**, 127–128 (2018).
50. Li, J., Tu, Y., Liu, R., Lu, Y. & Zhu, X. Toward "On-Demand" materials synthesis and scientific discovery through intelligent robots. *Adv. Sci*. **7**, 1901957 (2020).
51. Pells, R. Why scientists are delving into the virtual world. *Nature* https://doi.org/10.1038/d41586-023-02688-1 (2023).
52. Wang, G. et al. Development of metaverse for intelligent healthcare. *Nat. Mach. Intell*. **4**, 922–929 (2022).
53. Yoo, H. J. et al. Operation control system for task optimization and job parallelization via a user-optimal scheduler. https://doi.org/10.5281/zenodo.13990381 (2024).

Acknowledgements

This work was supported by the National Research Foundation of Korea funded by the Ministry of Science and ICT [NRF-2022M3H4A7046278 & RS-2024-00450102]. Nayeon Kim conceived the concept of image and drew Supplementary Fig. S1.

Author contributions

S.S.H., D.K., and K.Y.L. conceived the idea and supervised the project. H.J.Y. conceived the idea, designed OCTOPUS architecture, developed user-optimal scheduler, and performed experiments. All authors contributed to the result analysis and manuscript writing.

Competing interests

The authors declare no competing interests.

Additional information

Supplementary information The online version contains supplementary material available at https://doi.org/10.1038/s41467-024-54067-7.

Correspondence and requests for materials should be addressed to Kwan-Young Lee, Donghun Kim or Sang Soo Han.

Peer review information *Nature Communications* thanks the anonymous reviewer(s) for their contribution to the peer review of this work. A peer review file is available.