Enabling modular autonomous feedback-loops in materials science through hierarchical experimental laboratory automation and orchestration

Fuzhan Rahmanian[1,2], Jackson Flowers[1,2], Dan Guevarra[4], Matthias Richter[4], Maximilian Fichtner[1,3], John M. Gregoire[4,*], Helge S. Stein[1,2,*]

[1] Helmholtz Institute Ulm (HIU), Helmholtzstr. 11, 89081 Ulm, Germany
[2] Karlsruhe Institute of Technology (KIT), Institute of Physical Chemistry (IPC),Fritz-Haber-Weg 2, 76131 Karlsruhe, Germany
[3] Karlsruhe Institute of Technology (KIT), Institute of Nanotechnology (INT), P.O. Box 3640, D-76021 Karlsruhe, Germany
[4] Division of Engineering and Applied Science, and Liquid Sunlight Alliance (LiSA), California Institute of Technology (Caltech), Pasadena, California, USA

Address correspondence to: helge.stein@kit.edu, gregoire@caltech.edu

Abstract

Materials acceleration platforms (MAPs) operate on the paradigm of integrating combinatorial synthesis, high-throughput characterization, automatic analysis, and machine learning. Within these MAPs, one or multiple autonomous feedback loops may aim to optimize materials for certain functional properties or generate new insights. The scope of a given experiment campaign is defined by the range of experiment and analysis actions that are integrated into the experiment framework. Herein we present a method for integrating many actions within a hierarchical experimental laboratory automation and orchestration (HELAO) framework. We demonstrate the capability of orchestrating distributed research instruments that can incorporate data from experiments, simulations, and databases. HELAO interfaces laboratory hardware and software that are distributed across several computers and operating systems for executing experiments, data analysis, provenance tracking, and autonomous planning. Parallelization is an effective approach for accelerating knowledge generation provided that multiple instruments can be effectively coordinated, which we demonstrate with parallel electrochemistry experiments orchestrated by HELAO. Efficient implementation of autonomous research strategies requires device sharing, asynchronous multithreading, and

full integration of data management in experiment orchestration, which to the best of our knowledge, is demonstrated for the first time herein.

Introduction

Ever increasing performance demands necessitate the acceleration of materials science and chemistry.[1,2] Progress within the Materials Genome Initiative[3], high-throughput experimentation[4], and advances in machine learning, have enabled the emergent paradigm of conducting research in materials acceleration platforms (MAP)[5,6]. Within these MAPs different research tasks are accelerated and integrated to efficiently address the ever increasing complexity of materials optimization through targeted materials synthesis, processing, analysis and insight generation[7].

There have been limited demonstrations of autonomous workflows [8–11], but all have in common that the workflow is typically tied to a single instrument in a single laboratory[12]. This limited workflow complexity is rooted in the fact that most laboratory middleware orchestrating the actual laboratory hardware is designed around a single computer-instrument pairing[11]. Some notable examples include ChemOS[12] which in principle is capable of distributing work across different machines through the ROS[13] backend. This inarguably powerful software does however impose a complex overhead (ROS), a steep learning curve, and many (for most lab purposes unnecessary) dependencies. Commercial solutions to laboratory automation such as LabView by national Instruments are (initially) easier to deploy, but they typically come at a high-price, a subscription model, or they are tied to an operating system or lack some hardware/software support. Especially in instances where there is no (official) API for a device, or long-term instrument deployment with maintainability is necessary, both ROS and LabView based systems require significant adjustments during e.g. updates. Some framework designs like LabView make it increasingly hard to maintain automation software in an academic context with high turnover of staff and limit collaborative programming since source control for graphical programming is quite limited compared to text-based programming. As there are

some research fields like organic chemistry synthesis that require tailored languages[14] to express research tasks in a human and computer readable format[15,16] a high level of modularity is needed in conjunction with ontology informed design guidelines.

In addition to the necessity of being able to orchestrate a multitude of laboratory instruments, there is a critical need to be able to trace back all undertaken steps that lead to the acquisition of data or synthesis of a material [14,17] beyond FAIR[18] guidelines. Besides being FAIR compliant, there needs to be the possibility to re-run the exact same experiment from the “log” files[19] and all laboratory devices need to expose an interface to be run externally. If these criteria are met, autonomous inter-laboratory workflows [7] can be deployed. This however requires the atomic (as in lowest level possible) break down of high-level complex instructions like "measure electrochemistry at position #3". We therefore view the different levels of abstraction to be hierarchical in nature. The hierarchical laboratory automation and orchestration framework was built with the goals of being able to integrate any laboratory device for which a software driver is available or can be written, and to enable any configuration of the devices including serial and parallel experimentation, sharing of equipment across multiple instruments, and orchestration of multiple measurements in multiple laboratories. To facilitate continued adoption of active learning in experiment workflows, the framework is designed for facile switching between human and machine-based experiment selection. The framework adopts a data management wherein all gathered data and all instructions are stored in a FAIR format give the instruction data the same level of attention as the resulting measurement data. For these requirements to be met, HELAO needed to be able to communicate with devices within an instrument hosted or operated on different computers (i.e. some instruments are mutually exclusive to be connected on a PC due to driver constraints). We seek to be platform independent and require as little overhead as possible i.e. require minimal installation of additional software.

The presented hierarchical experimental laboratory automation and orchestration (HELAO) framework is building upon a widely deployed web framework called fastAPI[20] as shown in Figure 1. The main design idea is to represent every device of an instrument as a

(asynchronous) web server (Figure 1, right side). Basic functions of devices are exposed to and bundled by actions that themselves are again web servers. Only these actions are called by an orchestrator executing experiments on one or multiple instruments (Figure 1, left side). For future proofness code was developed in python 3.8+ with type hinting and pydantic type validation. The modular design allows for the integration of arbitrary devices, including those operated through OPC-UA[21].

The design guidelines and protocols necessary to orchestrate instrumentation in the laboratory are outlined in the following sections, together with a detailed description of the individual constituents. We demonstrate the orchestration of an active learning run on two instruments and deposited the resulting data including all instructions and the code at github.com/helgestein/helao-pub.

Methods

Design guidelines and protocols

From the bottom-up hardware perspective, a research instrument is made up from a set of devices. A device is a piece of laboratory equipment, defined as the largest "thing" that has a dedicated communication stream, i.e. a multi-channel potentiostat, or a motor control board. Devices are typically shipped with a driver that enables access to some or all its functions i.e. measuring a current. From the top-down perspective, a user or operator wishes to perform a series of experiments that are each a list of actionable events (which we define as "actions") that are to be executed in a particular order with predefined or variable parameters and/or designed on-the-fly via a decision policy. In this latter case, evaluation of the decision policy can be viewed as a particular type of action whose execution impacts future actions. The instructions for an experiment campaign are given to an orchestrator that governs their sequential execution from a queue, triggering actions in the appropriate order that are

materialized via the collection of drivers, thereby completing the top-to-bottom instrument framework. Everything that happens to or on that instrument originates within the orchestrator.

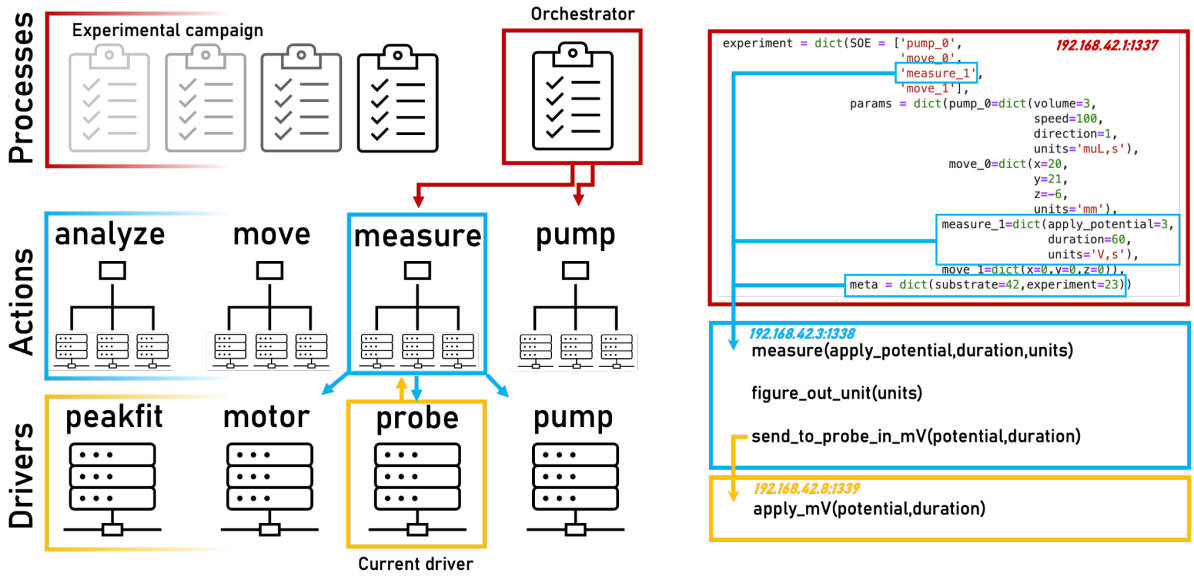

Figure 1. A schematic representation of HELAO where experiments are executed by sequentially calling actions which are high level wrappers for lower level driver instructions. Communication goes hierarchically down from the orchestrator level to actions, which may however communicate among each other, to the lowest level of drivers which can only communicate with actions. The orchestrator, actions, and drivers are all exposing python class functions through a web interface allowing for highly modular and distributed hosting of each item. Experiments are dictionaries containing a sequence of events (SOE) that outlines in which the actions are to be executed. Many experiments form a process. All actions require parameters and metadata that are all echoed back.

In order for the ensemble of devices to operate in concert as a single instrument, it is convenient to assemble the various elements listed above into a single software framework. Hierarchically from bottom to top, each device driver (internally communicating through e.g. serial, TCP/IP commands or a dynamic link library) is exposed through an uvicorn web server through fastAPI. Construction of more complicated but actionable functions ("actions") is based on these drivers exposed API calls.

An example of an action to pump a mixture of three fluids would therefore be initiated by the orchestrator calling the action to pump a mixture. This action then calls the pump driver server. Internally the driver server is sequentially called by the mixture action as the hardware requires to first initialize each pump channel, prime the pumps, and only then turn the pumps on. The

orchestrator will receive a nested reply from the action that entails all information exchanged, down to the lowest level i.e. initialization, priming, execution.

A rigorous commitment to data management is foundational to this framework's implementation. Requests to the driver and action servers track all functions called, as well as all (echoed) input parameters and outputs of those functions. The orchestrator tracks additional metadata, such as the time at which an action was performed or the point on a substrate at which an experiment was conducted, in addition to accepting arbitrary custom metadata. All of these are then automatically saved (redundantly), in the native file format (if applicable) and in a hdf5 file together with the parameters and metadata. Methods for depositing the hdf5 file into institutional repositories like Kadi4Mat[22] or MEAD[19] repositories are automatically executed after each session. From Kadi4Mat[22], experimental data can be accessed internally but also be shared with the community through materials cloud or AIIDA[23,24].

Due to each element of our software framework being a server, a very high degree of modularity is achieved. This allows, for instance, a single instance of a device to fulfill requests from multiple action servers sequentially. Precaution should however be taken as the current implementation of HELAO shows unstable behaviour when one action is being called by two independent orchestrators. A single orchestrator can however handle device sharing between multiple simultaneous experiments, as shown below. Another major benefit of this design is that it allows for distributed hosting of devices on different machines potentially scattered around the globe.

Drivers & Driver Servers

Any (autonomous) experimental workflow consists of smaller organizational units i.e. a scanning droplet cell (SDC)[25] instrument consists of several hardware devices like motors, a force sensor, pumps and a potentiostat. In addition there are software devices like e.g. analysis. All of these devices need to be able to receive commands, perform the instruction

and reply with measurement data or a status and echo back the input parameters. It should be mentioned that even an analysis as well as active learning script which do not physically “measure” something are implemented as a device. Each device is able to execute instructions from one or multiple instruments allowing for shared devices i.e. multi channel pumps connected to several instruments.

Driver Server design

Drivers only offer atomic interaction with devices i.e. the ability **to** connect and disconnect, return a status, read in a data stream (i.e. parsing proprietary file formats), and only execute rudimentary instructions (i.e. moving a motor one step). Some drivers are therefore more complex than others, as some devices offer direct python APIs whilst others require significant reverse engineering. Positively notable examples are for instance python drivers offered by Mecademic or Palmsens offering well documented SDKs. Each driver device pairing can only be a single server. Through these driver servers only (some) functions from the driver class can be called, to ensure a somewhat safe operation of the device, i.e. functions permanently altering a device behaviour are not callable by actions. Calling any driver functions can only be done through the web based API by sending web requests, enabling maximum modularity. This allows to mix and match devices in the software realm, just like screwing them together in the physical world.

Driver Parameters

Drivers accept parameters that are validated through pydantic data types that inherit from pydantic’s BaseModel. This automatically annotated and type hinted validation scheme allows users to assess how a request should be formatted in order to receive a desired device behaviour. Additionally, the pedantic validation scheme ensures proper data handling, easing

data management downstream. Traceability and ease in bug fixing are ensured by each driver server echoing all provided input parameters alongside the output data. For this purpose, the return object from any server including drivers is a dictionary containing two keys for the input parameters and output data. The parameter key is described by its name, the value(s), and optionally a physical unit. The data key contains data, which contains the data acquired or derived from the device. These dictionaries play a signal role between different organizational units, e.g. calling the pump, requires specification of volume, speed and direction. The response from the pump (a device acting but not measuring) is the entire serial string communication response (potentially containing valuable error messages) from the pump as output. The units returned for pumping are for instance speed in µL/min, total volume in µL and 1or 0 corresponding to a forward and backward pumping direction.

Actions & Action servers design

Hierarchically above drivers, actions wrap (several) driver functionalities towards more complex instructions. Similar to driver servers, actions also expose their functionalities as servers and again are not limited to a single instrument. Communication with multiple drivers is crucial as e.g. actuation and measurement of a resulting force in force limited movements is otherwise cumbersome to implement. Any communication between drivers on the driver level is forbidden and may only be invoked by an action.

Action parameters

Similar to driver servers, the return statement of the action servers consists of a dictionary containing two keys for the parameters and data. The output results from actions are the aggregate return statements received from all driver server calls downstream. After execution

of the relevant action function, the return statements of the called actions will be received by the orchestrator as the highest level of this hierarchy.

A major advantage of driver/action distinction is the possibility of multiple operating computers sharing one device. Any failures on a higher level (i.e. computer crash and/or program failure of the deployed visualizer or orchestrator) do not affect the operation of an instrument.

Another advantage of this design is the resolution of hardware conflicts and smart instrument communication, since some simultaneously executing actions could logically cause a contradiction. Therefore, a driver blocks further execution until the current request has been fulfilled. For instance, when the force sensor is measuring the amount of applied force as an action, this action will block further events until after execution. After the release, the next action will be called. This locked execution of sequential instructions allows for a safe operation without the need of a state machine. An alternative implementation would require a state machine on the highest level, thus violating the design principle that dictates little to no changes upon addition of new hardware.

Orchestrator / Local and External Database

The highest level in the framework is the orchestrator that sends out instructions to actions from a list of experiments (the process of a session) containing sequence of events and their accompanying parameters. The orchestrator is again a server that accepts experiments through an API function called addExperiment. This function adds an experiment to a list that is executed in the first-in-first-out order from the process, an infinite loop awaiting experiments. Upon exhaustion of experiments from the process the orchestrator remains online and awaits the next experiment(s). Booting up the HELAO framework for an instrument requires launching of minimally n*2+1 servers, where n is the number of devices in the system. The booting sequence is done through a launch script with a simple user interface requiring nothing but an

instrument specific configuration file. Thereby, the only step necessary to launch an instrument is setting a path to a file containing all parameters. If two copies of an instrument exist, they require unique configuration files with unique IP addresses. Therefore the process of cloning an instrument is reduced to replacing IP addresses.

Though the orchestrator sits on the highest level of the hierarchy, it may only access the exposed GET and POST routines from action servers.

Defining a process

The main purpose of the orchestrator is the execution of a list of (dynamically editable) experiments from the process as well as data management. For defining a sequential experiment involving multiple devices, experiments with sequence of events and parameters need to be defined. An instrument is factually defined by the devices called in a sequence of events. An experiment is defined by a dictionary of two keys. The *Sequence of Events* (SOE) key contains an ordered list outlining the exact order in which actions are to be executed. The “params” key contains all necessary parameters for any of the SOE actions. As actions may be called multiple times they are numbered sequentially in the SOE. Calling an action through the orchestrator requires four parts: the name of the action, the desired function, a number which indicates the n-th time that we call that specific action within a SOE and the thread number (e.g. "motor/moveAbs-3:1" for calling the third absolute movement of a motor belonging to thread 1). Thread numbers are important for deploying active learning across multiple instruments.

The parameters for actions are stored under the "MoveAbs-3" key in the parameter key and contains the specific values for running that particular action (i.e. dx = 2 mm ,dy = 3 mm, dz = 0 mm).

Defining a session

To start and end a session lasting for one or more experiments the first and last actions to be called are the “start” and “finish” actions, natively implemented within the orchestrator (i.e. not as servers), hence being called native actions. The data acquired within a session is locally stored in a single hdf5 file that is then uploaded to KaDI4Mat upon calling the “finish” action. Storing data locally and uploading it at the end of a session has been shown to be significantly faster and less error prone.

Data analysis and machine learning servers

A goal of HELAO is to enable active learning accelerated experiments across a wide range of laboratory instruments. Active learning does however require automatic data analysis and machine learning based suggestion of the next best subsequent experiment.

These two functionalities require servers in HELAO providing these functionalities. An active learning action within a SOE requires access to the data of one or all previous actions and the possibility to alter a parameter for a future action of the current SOE to be executed. The data analysis servers or actions, act like any other action/driver server combination with the exception that they provide pointers to the location in the orchestrator memory of where relevant data is stored. Likewise, the server dedicated to machine learning for active learning needs to be pointed to the input values of the experiments and the resulting target values from data analysis. Inside the active learning action, the datasets are aggregated on the fly from the orchestrator temporary storage (what is later the hdf5 file being uploaded). A suggestion can be made from a list of candidate parameters or be freely decided by the algorithm depending on the chosen optimizer. This active learning action then passes the suggested parameter pair through a special native command pointing to this “source value” returned by

the action. This special native action is called after the active learning action and before the to be modified action. The active learning suggested parameter is then copied over to the “target” parameter of the subsequent experiment.

These functionalities allow for autonomous operation where the user only has to define the budget of active learning runs, pointers to the input and output values, and the choice of optimizer and the estimator.

The active learning server contains a broad range of optimizers and regression algorithms. Also, several acquisition functions have been implemented including expected improvement (EI), probability of improvement (POI).

As some ML algorithms require significant computational resources within a fastAPI thread and some actions are data transfer intensive, servers may become unresponsive. To solve this issue, the most computationally expensive tasks like machine learning can be wrapped inside a celery server when necessary that is capable of distributing high workloads across compute clusters. We empirically observed this necessity for long running active learning runs with a high degree of freedom.

Visualizer

On the same hierarchical level of the orchestrator is the visualizer, which can be viewed as a “read only” orchestrator that has global access and does not store data. This server can display the live data of e.g. electrochemical test measurements or Raman spectroscopy to assess data quality during a run.

Results

Implementation of hardware drivers

The aim of HELAO is to be a universal laboratory automation framework, democratizing accelerated experimental research workflows. To this end, the two laboratories at the California Institute of Technology (Caltech) and Karlsruhe Institute of Technology (KIT) started to implement major hardware components amenable for automatization. In Table 1, all currently implemented hardware drivers are listed. During development of these drivers, it became apparent that there exist two major types of drivers, those with blocking and non-blocking operation. Those with non-blocking operations typically accept an instruction, execute it, and require the user to ask if the current operation is finished.

A special class of devices are auxiliary (aux) devices. These are broadly defined as software "devices" used e.g. for data analysis, regression, and prediction. These seemingly simple drivers may be of equal complexity as HELAO with examples including powerful background inference agorithms[26] and multimodel active learning[27] implemented through a load distributing celery server as described in the driver section.

Generally speaking, with the devices available at the time of writing this manuscript, highly complex instruments have and are being built, whose detailed descriptions will be the subject of future work. As an initial example of the scope of the present HELAO implementations, the operated instruments are comprised of the devices shown in Table 1 that include four scanning droplet cells at KIT *(each consisting of lang, autolab, pump, force, aux, kadi)*, one scanning droplet cell at Caltech *(galil, gamry)*, a coupled Raman and FTIR spectrometer *(owis, ocean, arcoptix,aux, kadi)*, a battery cycler *(arbin, aux)*, and a coin cell assembly system *(mecademic, rail, arbin, arduino, aux)*.

Device Name	Type	Communicatio	Measures/Cont	Manufacturer	natively

		n	rols		blocking
lang	Motion	.net API	position	Lang GmbH	no
galil	Motion, IO	TCP/IP	position	Galil Motion Control Inc.	no
owis	Motion	serial commands	position	Owis GmbH	no
mecademic	Motion	python TCP/IP API	position, state	Mecademic Ltd.	no
rail	Motion	TCP/IP	position	Jenny Science AG	no
autolab	Potentiostat	.net API	electrochemistry	Methrohm Autolab B.V.	yes
gamry	Potentiostat	.dll for serial communication	electrochemistry	Gamry Instruments Inc.	yes
arbin	Potentiostat	autohotkey	electrochemistry	Arbin Inc.	no
pump	pumping	serial commands	n.a.	CAT engineering GmbH	no
arcoptix	spectroscopy	.dll api	IR spectra	arcoptix S.A.	yes
ocean	spectroscopy Raman	python package	Raman spectra	ocean insights GmbH	yes
force	force sensing	serial commands	force	ME Meßsysteme GmbH	n/a
arduino	relays, I/O	python package	n.a.	arduino	no
kadi	data management	python package	n.a.	KIT	yes
aux	machine learning and analysis	python package	n.a.	n.a.	yes

Table 1: Currently implemented devices in the laboratories at KIT and Caltech. Instruments built from this include scanning droplet cells, high-throughput spectrometers and a battery assembly robot. The extreme modularity allows to mix and match any of these devices by simply defining a sequence of

events i.e. to build an integrated SDC and spectrometer or a sample exchange robot without code changes to HELAO. For each device we note the communication protocol and the physical quantity being controlled and/or measurement. We also note whether the instrument is "natively blocking" meaning that the device is unable to process new commands until the currently running command is finished.

Hardware in the loop active learning

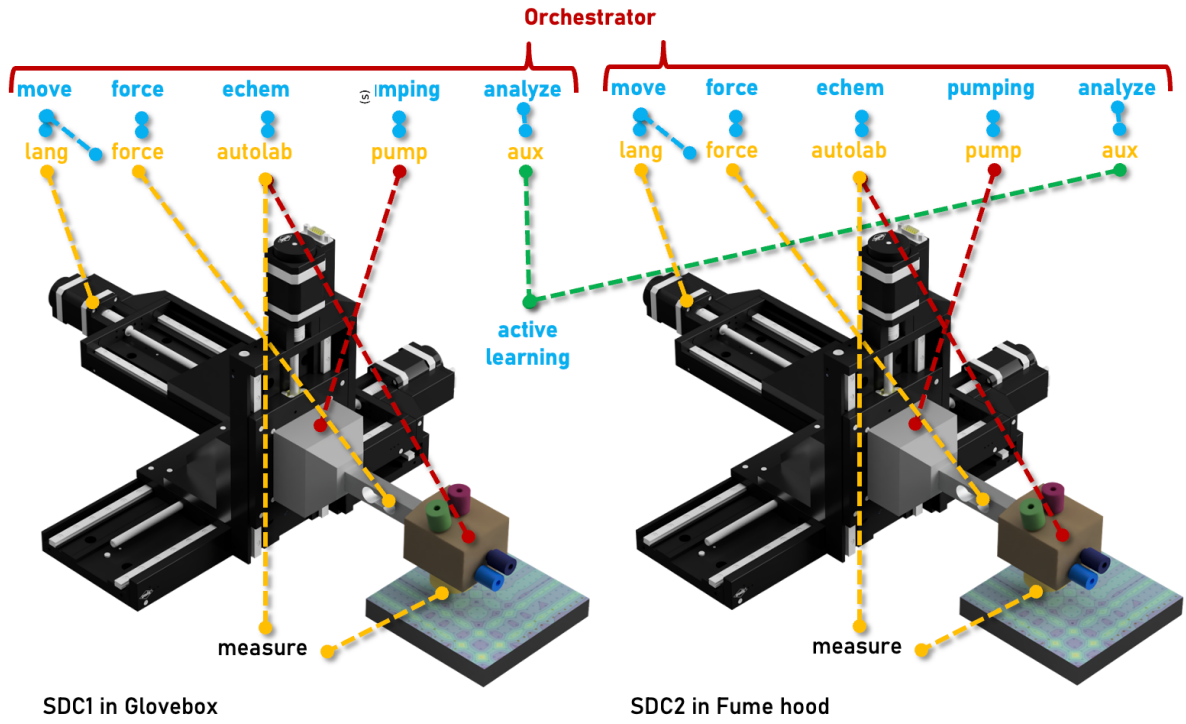

Figure 2. Schematic drawing of the HELAO hardware in the loop active learning run with two instruments running parallel and the corresponding actions, drivers and orchestrator. Compared to two SDCs running in parallel this instrument has no pump and the potentiostat (shown as red dashed lines) is replaced by a measure driver that returns a function value from the schwefel function depending on the visited substrate position. The active learning action and driver are shared among the instruments/threads.

A hardware in the loop demonstration run of HELAO is shown in Fig. 2. The instrument is copied two times where one setup was run in a fume hood and another one was run in a glove box (hosting the orchestrator). The learner and optimizer is the same for both instruments as it is centrally run by the orchestrator. Compared to a standard SDC the only difference is, that there is no potentiostat and no pump used. This is to point the attention to the (arguably

already complex) software hardware interaction and not to the underlying electrochemistry. An example video of a parallel active learning run can be found in the supplemental information. Replacing the potentiostat in this hardware in the loop demonstration is done through the implementation of a device called measure which is analyzed by a dummy analysis . This device returns a scaled Schwefel function[28] depending on the position where the SDC touches down on a substrate.

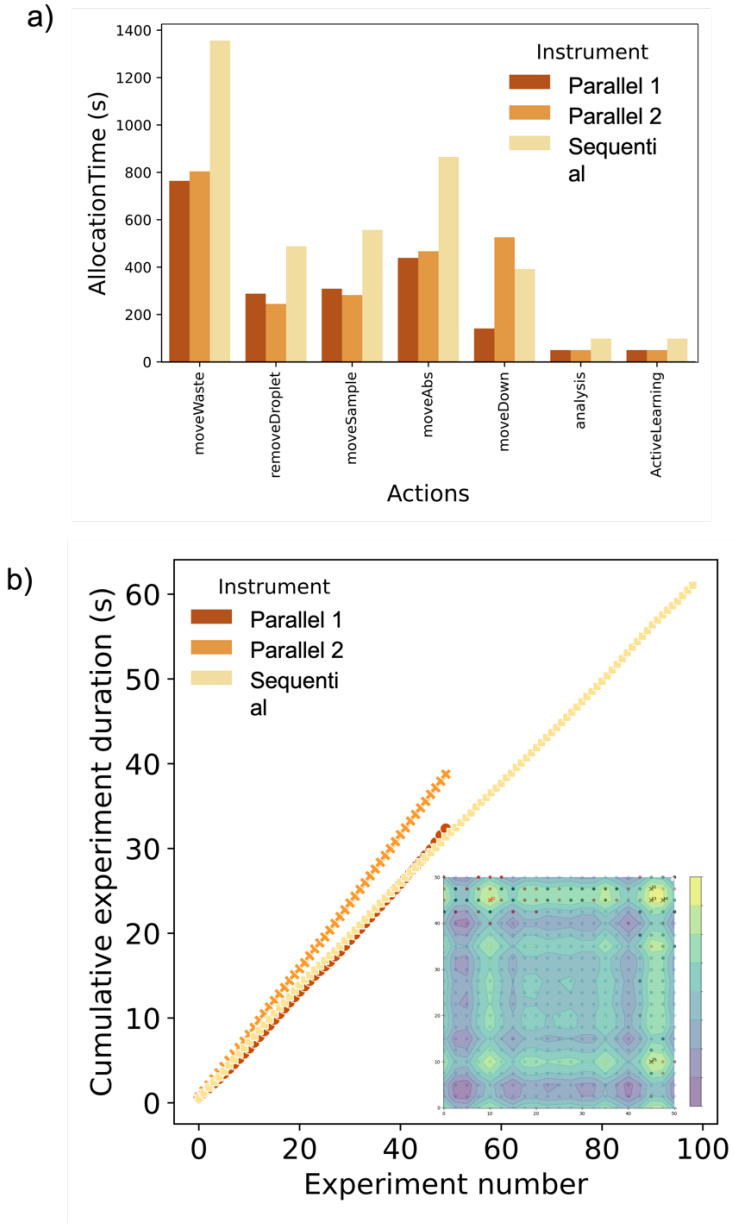

Figure 3: a) Time spent at each action for a sequential and a parallel run with two instruments b) total time spent per run. The time spent does not form a perfectly straight line as some actions need different time (i.e. movements are shorter or longer). The inset in c shows the parallel run and highlights visited points in black and red depending on whether they were visited by instrument 1 or 2. The sequence of events is roughly the order shown in a).

The active learning run is stopped once a threshold value (top percentile) is found. Actions in this run consist of e.g. “move to waste”, “remove the droplet”, “move to sample offset”, “move to the defined point”, “move down to substrate”, “get output value”, “predict the next best position using active learning algorithm”. One experiment takes about 108 seconds. Depending on the number of datapoints the learning step requires more time. During the measurement, all data is constantly logged from all devices and subsequently uploaded to the data management repository (kadi4mat). The overall time required for the entire run was a little less than 3 hours and allows for a fine grained analysis of what action consumes the most experimental time as shown in Figure 2b. From this analysis, it is for instance evident that motor movement between measurements consumes a substantial fraction of the experimental time, motivating efforts to enable faster movement. With 3856 seconds for the sequential run there is a significant speed up when the experiment is run in parallel where the instrument 1 (in the fume hood) takes 2041 and the instrument 2 requires 2424 seconds to complete. As is evident from these numbers the speedup is a little less than 2x for a parallel active learning run as asynchronous locks and the machine learning consume some of the time. To the best of our knowledge this is the first demonstration of an active learning run involving two spatially distributed instruments involving more than one operational PC. The hdf5 file generated during this run was recorded on 05.10.2021 and has been uploaded to KaDI4Mat upon completion of the session under the records 20287 and 20280. Public release of the dataset with the DOI XXX has been triggered on 09.10.2021. The hdf5 file for this run may also be found in the supplementary information.

Discussion

Herein we present a versatile, stable and modular approach to laboratory automatization that offers capabilities to deploy autonomous experimentation in materials science. The framework was built using modern asynchronous programming and operates in a safe hierarchical layout. State of the art server-based communication between laboratory devices is used to ensure

maximum modularity and reusability of devices across instruments and laboratories. Higher level sequences requiring the interaction with one or between several devices are wrapped in actions that are exposed as web servers. This design allows for a distributed operation across computers and locations, in addition to being resilient against single machine crashes.

Through utilization of a facile underlying web framework like fastAPI and pydantic type annotation documentation to most functions is autogenerated. This design allows users also to quickly adopt new devices and actions without the need of installing clients or servers, as drivers and actions can be called through python's built in 'request' package or even through the auto generated web documentation. Moreover this allows any HELAO action driver combo to be called by virtually any other software if users wish to implement their own flavour of an orchestrator. If users wish to deploy active learning to a device that does not accept standard web requests like OPC-UA (often encountered in industrial settings) a simple deployment of a wrapper between OPC-UA and fastAPI can be used. As there is just one function in the HELAO code that does the actual requests to servers hosting actions the current design even allows printing the code over to other communication protocols with little code changes.

This high degree of modularity and interoperability is only possible through a very lightweight design that puts very little restrictions on the user contrary to complex middleware like ROS or ChemOS. Other competing softwares and frameworks are ARES OS that is currently only demonstrated to operate on a single computer instrument pairing. Another mature and great alternative to the lightweight implementation of HELAO is the bluesky project[29]. Bluesky works with similar hardware abstraction ideas like HELAO, but puts significantly more constraints on a user and is, in our view, more built around streamlining the research process as a whole. However, orchestrating multiple instruments in parallel has not been demonstrated by any other laboratory automation framework. These parallelization efforts are however necessary for deploying state of the art optimizers that incorporate uncertainty and different optimization strategies like GOLEM[30].

The framework is built with the goal of being fully FAIR compliant and allows users to rerun an experiment without much or any overhead. We view this degree of data management to be

FAIR+. By logging every possible parameter along the entire research process, it is possible to extract utilization figures, find bugs, and determine bottlenecks in high-throughput experimentation. Direct interfacing with data management software has been demonstrated, to the best of our knowledge, for the first time in an autonomous research environment. All data gathered during the active learning sessions has been automatically uploaded upon the completion of the session and is publicly available at figshare[31] and from the SI of this manuscript. Within the university network all recorded data is made publicly available by default without an embargo period as a statement to encourage more data sharing. HELAO is demonstrated to be stable and versatile and is published under the LGPL license at https://github.com/helgestein/helao-dev. Stand-alone example configurations with reference driver implementations are available alongside a documentation thereof as part of the repository.

The parallel active learning run with hardware in the loop of HELAO demonstrates for the first time that two (and technically unlimited more) spatially apart instruments in a materials science laboratory are capable of collaboratively optimizing together for faster discovery. Contributions and collaborations with and by the community to expand the hardware support for HELAO is therefore warmly welcome. Future efforts will aim to bridge HELAO with methods from theoretical materials science to build modular physics-informed instrumentation and autonomous feedback loops connecting laboratories.

Acknowledgements

This work contributes to the research performed at CELEST (Center for Electrochemical Energy Storage Ulm-Karlsruhe) and was partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany´s Excellence Strategy – EXC 2154 – Project number 390874152. This project has also received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957189. Design of software architecture at Caltech was supported by the Liquid Sunlight Alliance, which is supported by the U.S. Department of Energy (DOE), Office of

Science, Office of Basic Energy Sciences (BES), Fuels from Sunlight Hub under Award Number DE-SC0021266.

Author contributions statement

H.S.S. and J.M.G conceived the idea and designed the first software layout. H.S.S. developed the first drivers and server based communication protocols. F.R., J.F., D.G., and M.R. implemented drivers, wrote actions and conducted the experiments. F.R. implemented drivers pertaining to SDC and deployed machine learning algorithms to HELAO. J.F. implemented the majority of functions within the orchestrator. All authors reviewed the manuscript.

(1) Alberi, K.; Nardelli, M. B.; Zakutayev, A.; Mitas, L.; Curtarolo, S.; Jain, A.; Fornari, M.; Marzari, N.; Takeuchi, I.; Green, M. L.; Kanatzidis, M.; Toney, M. F.; Butenko, S.; Meredig, B.; Lany, S.; Kattner, U.; Davydov, A.; Toberer, E. S.; Stevanovic, V.; Walsh, A.; Park, N.-G.; Aspuru-Guzik, A.; Tabor, D. P.; Nelson, J.; Murphy, J.; Setlur, A.; Gregoire, J.; Li, H.; Xiao, R.; Ludwig, A.; Martin, L. W.; Rappe, A. M.; Wei, S.-H.; Perkins, J. The 2019 Materials by Design Roadmap. *J. Phys. Appl. Phys.* **2019**, *52* (1), 013001. https://doi.org/10.1088/1361-6463/aad926.

(2) Correa-Baena, J.-P.; Hippalgaonkar, K.; van Duren, J.; Jaffer, S.; Chandrasekhar, V. R.; Stevanovic, V.; Wadia, C.; Guha, S.; Buonassisi, T. Accelerating Materials Development via Automation, Machine Learning, and High-Performance Computing. *Joule* **2018**, *2* (8), 1410–1420. https://doi.org/10.1016/j.joule.2018.05.009.

(3) Green, M. L.; Choi, C. L.; Hattrick-Simpers, J. R.; Joshi, A. M.; Takeuchi, I.; Barron, S. C.; Campo, E.; Chiang, T.; Empedocles, S.; Gregoire, J. M.; Kusne, A. G.; Martin, J.; Mehta, A.; Persson, K.; Trautt, Z.; Van Duren, J.; Zakutayev, A. Fulfilling the Promise of the Materials Genome Initiative with High-Throughput Experimental Methodologies. *Appl. Phys. Rev.* **2017**, *4* (1), 11105. https://doi.org/10.1063/1.4977487.

(4) Amis, E. J.; Xiang, X. D.; Zhao, J. C. Combinatorial Materials Science: What's New since Edison? *MRS Bull.* **2002**.

(5) Materials Acceleration Platform—Accelerating Advanced Energy Materials Discovery by Integrating High-Throughput Methods with Artificial Intelligence. 109.

(6) Aykol, M.; Hummelshøj, J. S.; Anapolsky, A.; Aoyagi, K.; Bazant, M. Z.; Bligaard, T.; Braatz, R. D.; Broderick, S.; Cogswell, D.; Dagdelen, J.; Drisdell, W.; Garcia, E.; Garikipati, K.; Gavini, V.; Gent, W. E.; Giordano, L.; Gomes, C. P.; Gomez-Bombarelli, R.; Balaji Gopal, C.; Gregoire, J. M.; Grossman, J. C.; Herring, P.; Hung, L.; Jaramillo, T. F.; King, L.; Kwon, H.-K.; Maekawa, R.; Minor, A. M.; Montoya, J. H.; Mueller, T.; Ophus, C.; Rajan, K.; Ramprasad, R.; Rohr, B.; Schweigert, D.; Shao-Horn, Y.; Suga, Y.; Suram, S. K.; Viswanathan, V.; Whitacre, J. F.; Willard, A. P.; Wodo, O.; Wolverton, C.; Storey, B. D. The Materials Research Platform: Defining the Requirements from User Stories. *Matter* **2019**, *1* (6), 1433–1438. https://doi.org/10.1016/j.matt.2019.10.024.

(7) Stein, H. S.; Gregoire, J. M. Progress and Prospects for Accelerating Materials Science with Automated and Autonomous Workflows. *Chem. Sci.* **2019**, *10* (42), 9640–9649. https://doi.org/10.1039/C9SC03766G.

(8) Jensen, K. F.; Coley, C. W.; Eyke, N. S. Autonomous Discovery in the Chemical Sciences Part I: Progress. *Angew. Chem. Int. Ed. n/a* (n/a). https://doi.org/10.1002/anie.201909987.

(9) Coley, C. W.; Eyke, N. S.; Jensen, K. F. Autonomous Discovery in the Chemical Sciences Part II: Outlook. *Angew. Chem. Int. Ed.* **2019**. https://doi.org/10.1002/anie.201909989.

(10) Dimitrov, T.; Kreisbeck, C.; Becker, J. S.; Aspuru-Guzik, A.; Saikin, S. K. Autonomous Molecular Design: Then and Now. *ACS Appl. Mater. Interfaces* **2019**. https://doi.org/10.1021/acsami.9b01226.

(11) Nikolaev, P.; Hooper, D.; Webber, F.; Rao, R.; Decker, K.; Krein, M.; Poleski, J.; Barto, R.; Maruyama, B. Autonomy in Materials Research: A Case Study in Carbon

Nanotube Growth. *Npj Comput. Mater.* **2016**, *2*, 16031. https://doi.org/10.1038/npjcompumats.2016.31.

(12) Roch, L. M.; Häse, F.; Kreisbeck, C.; Tamayo-Mendoza, T.; Yunker, L. P. E.; Hein, J. E.; Aspuru-Guzik, A. ChemOS: Orchestrating Autonomous Experimentation. *Sci. Robot.* **2018**, *3* (19). https://doi.org/10.1126/scirobotics.aat5559.

(13) Quigley, M.; Gerkey, B.; Conley, K.; Faust, J.; Foote, T.; Leibs, J.; Berger, E.; Wheeler, R.; Ng, A. ROS: An Open-Source Robot Operating System. 6.

(14) Pendleton, I. M.; Cattabriga, G.; Li, Z.; Najeeb, M. A.; Friedler, S. A.; Norquist, A. J.; Chan, E. M.; Schrier, J. Experiment Specification, Capture and Laboratory Automation Technology (ESCALATE): A Software Pipeline for Automated Chemical Experimentation and Data Management. *MRS Commun.* **2019**, *9* (3), 846–859. https://doi.org/10.1557/mrc.2019.72.

(15) Gromski, P. S.; Granda, J. M.; Cronin, L. Universal Chemical Synthesis and Discovery with 'The Chemputer.' *Trends Chem.* **2020**, *2* (1), 4–12. https://doi.org/10.1016/j.trechm.2019.07.004.

(16) Gromski, P. S.; Henson, A. B.; Granda, J. M.; Cronin, L. How to Explore Chemical Space Using Algorithms and Automation. *Nat. Rev. Chem.* **2019**, 1. https://doi.org/10.1038/s41570-018-0066-y.

(17) Castelli, I. E.; Arismendi-Arrieta, D. J.; Bhowmik, A.; Cekic-Laskovic, I.; Clark, S.; Dominko, R.; Flores, E.; Flowers, J.; Frederiksen, K. U.; Friis, J.; Grimaud, A.; Hansen, K. V.; Hardwick, L. J.; Hermansson, K.; Königer, L.; Lauritzen, H.; Cras, F. L.; Li, H.; Lyonnard, S.; Lorrmann, H.; Marzari, N.; Niedzicki, L.; Pizzi, G.; Rahmanian, F.; Stein, H.; Uhrin, M.; Wenzel, W.; Winter, M.; Wölke, C.; Vegge, T. Data Management Plans: The Importance of Data Management in the BIG-MAP Project. *ArXiv210601616 Cond-Mat* **2021**.

(18) Wilkinson, M. D.; Dumontier, M.; Aalbersberg, Ij. J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.-W.; da Silva Santos, L. B.; Bourne, P. E.; Bouwman, J.; Brookes, A. J.; Clark, T.; Crosas, M.; Dillo, I.; Dumon, O.; Edmunds, S.; Evelo, C. T.;

Finkers, R.; Gonzalez-Beltran, A.; Gray, A. J. G.; Groth, P.; Goble, C.; Grethe, J. S.; Heringa, J.; t Hoen, P. A. C.; Hooft, R.; Kuhn, T.; Kok, R.; Kok, J.; Lusher, S. J.; Martone, M. E.; Mons, A.; Packer, A. L.; Persson, B.; Rocca-Serra, P.; Roos, M.; van Schaik, R.; Sansone, S.-A.; Schultes, E.; Sengstag, T.; Slater, T.; Strawn, G.; Swertz, M. A.; Thompson, M.; van der Lei, J.; van Mulligen, E.; Velterop, J.; Waagmeester, A.; Wittenburg, P.; Wolstencroft, K.; Zhao, J.; Mons, B. The FAIR Guiding Principles for Scientific Data Management and Stewardship. *Sci. Data* **2016**, *3*, 160018–160019. https://doi.org/10.1038/sdata.2016.18.

(19) Soedarmadji, E.; Stein, H. S.; Suram, S. K.; Guevarra, D.; Gregoire, J. M. Tracking Materials Science Data Lineage to Manage Millions of Materials Experiments and Analyses. *Npj Comput. Mater.* **2019**, *5* (1), 79. https://doi.org/10.1038/s41524-019-0216-x.

(20) FastAPI https://fastapi.tiangolo.com/#license (accessed 2021 -06 -21).

(21) Home Page https://opcfoundation.org/ (accessed 2021 -06 -21).

(22) Brandt, N.; Griem, L.; Herrmann, C.; Schoof, E.; Tosato, G.; Zhao, Y.; Zschumme, P.; Selzer, M. Kadi4Mat: A Research Data Infrastructure for Materials Science. *Data Sci. J.* **2021**, *20*, 8. https://doi.org/10.5334/dsj-2021-008.

(23) Talirz, L.; Kumbhar, S.; Passaro, E.; Yakutovich, A. V.; Granata, V.; Gargiulo, F.; Borelli, M.; Uhrin, M.; Huber, S. P.; Zoupanos, S.; Adorf, C. S.; Andersen, C. W.; Schütt, O.; Pignedoli, C. A.; Passerone, D.; VandeVondele, J.; Schulthess, T. C.; Smit, B.; Pizzi, G.; Marzari, N. Materials Cloud, a Platform for Open Computational Science. *Sci. Data* **2020**, *7* (1), 299. https://doi.org/10.1038/s41597-020-00637-5.

(24) Pizzi, G.; Cepellotti, A.; Sabatini, R.; Marzari, N.; Kozinsky, B. AiiDA: Automated Interactive Infrastructure and Database for Computational Science. *Comput. Mater. Sci.* **2016**, *111*, 218–230. https://doi.org/10.1016/j.commatsci.2015.09.013.

(25) The potential of scanning electrochemical probe microscopy and scanning droplet cells in battery research - Daboss - - Electrochemical Science Advances - Wiley Online Library https://chemistry-

europe.onlinelibrary.wiley.com/doi/full/10.1002/elsa.202100122 (accessed 2021 -10 -08).

(26) Ament, S. E.; Stein, H. S.; Guevarra, D.; Zhou, L.; Haber, J. A.; Boyd, D. A.; Umehara, M.; Gregoire, J. M.; Gomes, C. P. Multi-Component Background Learning Automates Signal Detection for Spectroscopic Data. *Npj Comput. Mater.* **2019**, *5* (1), 1–7. https://doi.org/10.1038/s41524-019-0213-0.

(27) Rohr, B.; Stein, H. S.; Guevarra, D.; Wang, Y.; Haber, J. A.; Aykol, M.; Suram, S. K.; Gregoire, J. M. Benchmarking the Acceleration of Materials Discovery by Sequential Learning. *Chem. Sci.* **2020**, *11* (10), 2696–2706. https://doi.org/10.1039/C9SC05999G.

(28) Häse, F.; Aldeghi, M.; Hickman, R. J.; Roch, L. M.; Christensen, M.; Liles, E.; Hein, J. E.; Aspuru-Guzik, A. Olympus: A Benchmarking Framework for Noisy Optimization and Experiment Planning. *ArXiv201004153 Phys. Stat* **2020**.

(29) Allan, D.; Caswell, T.; Campbell, S.; Rakitin, M.; (Bluesky's Ahead: A Multi-Facility Collaboration for an a la Carte Software Project for Data Acquisition and Management, Synchrotron Radiation News, **2019**. 32:3, 19-22, DOI: 10.1080/08940886.2019.1608121

(30) Aldeghi M.; F. Häse; R. J. Hickman; I. Tamblyn;A Aspuru-Guzik, Golem: An algorithm for robust experiment and process optimization *ArXiv*2103.03716 *math. OC* **2021**

(31) HDF5 Files DOI: 10.6084/m9.figshare.16798177.v1 https://figshare.com/s/1578223bbf5ddde605af