Autonomous Robots (2023) 47:1057–1086
https://doi.org/10.1007/s10514-023-10136-2

Large language models for chemistry robotics

Naruki Yoshikawa[1,2] · Marta Skreta[1,2] · Kourosh Darvish[1,2] · Sebastian Arellano-Rubach[3] · Zhi Ji[1] · Lasse Bjørn Kristensen[1] · Andrew Zou Li[1] · Yuchi Zhao[4] · Haoping Xu[1,2] · Artur Kuramshin[1] · Alán Aspuru-Guzik[1,2,5] · Florian Shkurti[1,2] · Animesh Garg[1,2,6]

Published online: 25 October 2023

Abstract
This paper proposes an approach to automate chemistry experiments using robots by translating natural language instructions into robot-executable plans, using large language models together with task and motion planning. Adding natural language interfaces to autonomous chemistry experiment systems lowers the barrier to using complicated robotics systems and increases utility for non-expert users, but translating natural language experiment descriptions from users into low-level robotics languages is nontrivial. Furthermore, while recent advances have used large language models to generate task plans, reliably executing those plans in the real world by an embodied agent remains challenging. To enable autonomous chemistry experiments and alleviate the workload of chemists, robots must interpret natural language commands, perceive the workspace, autonomously plan multi-step actions and motions, consider safety precautions, and interact with various laboratory equipment. Our approach, CLAIRIFY, combines automatic iterative prompting with program verification to ensure syntactically valid programs in a data-scarce domain-specific language that incorporates environmental constraints. The generated plan is executed through solving a constrained task and motion planning problem using PDDLStream solvers to prevent spillages of liquids as well as collisions in chemistry labs. We demonstrate the effectiveness of our approach in planning chemistry experiments, with plans successfully executed on a real robot using a repertoire of robot skills and lab tools. Specifically, we showcase the utility of our framework in pouring skills for various materials and two fundamental chemical experiments for materials synthesis: solubility and recrystallization. Further details about CLAIRIFY can be found at https://ac-rad.github.io/clairify/.

Keywords Large language models · Constrained task and motion planning · Plan generation verification · Self-driving labs · Chemistry lab automation

1 Introduction

The execution of chemistry experiments, which represents a crucial stage in the process of material discovery, typically relies on human experts. This manual experimentation poses a number of significant challenges, such as difficulties in reproducibility, high resource requirements, and limited scalability. To address these obstacles, the concept of self-driving labs (SDLs) has emerged (Seifrid et al., 2022). Although specialized hardware for chemistry experiments has been proposed and used in modern labs, we argue that using general-purpose robots is beneficial in developing SDLs that maximize the use of existing resources and are more configurable. The functionality of general-purpose robots can be expanded by programming them to interact with existing chemistry instruments designed for humans. This feature contributes to reducing the cost related to introducing addi-

Naruki Yoshikawa, Marta Skreta and Kourosh Darvish have contributed equally to this work.

✉ Kourosh Darvish
kdarvish@cs.toronto.edu

1 University of Toronto, Toronto, ON, Canada
2 Vector Institute for Artificial Intelligence, Toronto, ON, Canada
3 University of Toronto Schools, Toronto, ON, Canada
4 University of Waterloo, Waterloo, ON, Canada
5 CIFAR Artificial Intelligence Research Chair, Toronto, ON, Canada
6 NVIDIA, Santa Clara, CA, USA

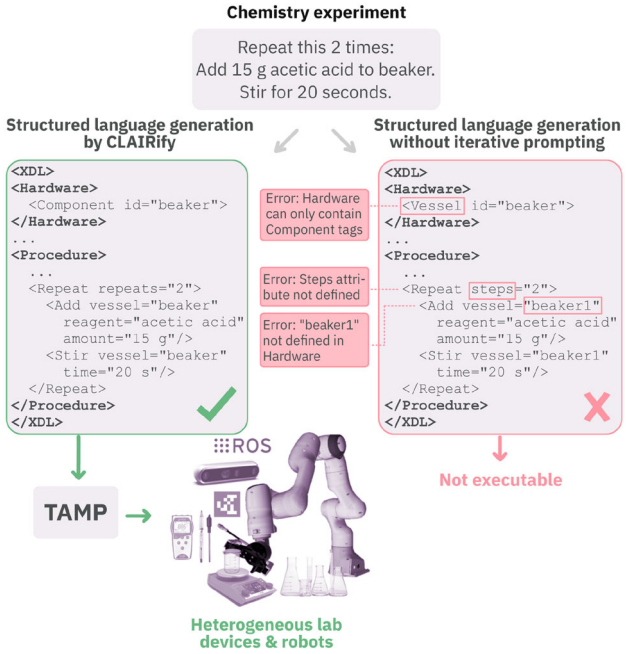


Fig. 1 Task plans generated by LLMs may contain syntactical errors in domain-specific languages. By using verifier-assisted iterative prompting, CLAIRIFY can generate a valid program. Once the program has been verified, it is passed on to a task and motion planner (TAMP) for execution by a robot

tional hardware for specific purposes. Therefore, this study discusses the use of general-purpose robots for building chemistry SDLs.

One of the principal obstacles in effectively using robots in chemistry labs is to ensure that they are natural and intuitive for chemists to operate. An approach to achieving a natural and intuitive interface between chemists and robots is through the use of natural language as a communication medium. This approach enables users to instruct robots in an efficient and effective manner.

This work aims to facilitate autonomous and safe execution of chemistry experiments using general-purpose robot manipulators. This is accomplished through natural language instructions to generate plans. Several challenges must be addressed at both the natural language processing (NLP) and robotic planning levels for this purpose. At the NLP level, the robot must be capable of converting natural language instructions into executable robot instructions (Fig. 1). At the robotic planning level, the robotic system should be capable of planning robot tasks and motions that take safety considerations into account, using intermediate goals identified by NLP and perceptual information of the robot workspace.

Natural language has been used in the literature to overcome the communication barrier between humans and robots, for example in navigation tasks (Tellex et al., 2011). More recently, numerous studies have demonstrated that large language models (LLMs) can assist robots to reason with common sense (Huang et al., 2022; Singh et al., 2022). LLMs have been used to generate structured outputs (Devlin et al., 2019; Brown et al., 2020; Chowdhery et al., 2022), including code generation (Chen et al., 2021; Wang et al., 2021; Li et al., 2022) and robot programming (Liang et al., 2022). Nevertheless, the application of LLMs to task-plan generation for chemistry robots presents two key challenges. Just like with all machine-executable code, generated plans must adhere to strict syntax rules in order to be executed by a robot. In robotics, task-plan verification is often desired, because it increases the likelihood that a robot can reach the desired goal (Garrett et al., 2020; Ahn et al., 2022). Furthermore, LLMs may perform poorly in generating task plans in a zero-shot manner for data-scarce, domain-specific languages, such as the ones in robot planning, and thus require fine-tuning, or in technical scientific domains (Gu et al., 2021), or scientific coding (Liu et al., 2023). Several approaches have been proposed in the literature to address these issues. One promising technique is iterative prompting, which has an advantage over fine-tuning LLMs, as the latter requires access to the model weights and training datasets to learn domain-specific languages reasonably well and incurs high computational costs (Mishra et al., 2021; Wang et al., 2011; Wu et al., 2022). Iterative prompting enables the LLM to verify candidate plans while providing the rules of structured language as input, thereby leveraging in-context learning.

The planning component of the robotic system takes as input perception, vision-based outcome evaluation of experiments and natural language instructions, and solves a constrained task and motion planning (TAMP) problem. To do so, the robot must possess both general and chemistry domain-specific perception and manipulation skills, including recognizing transparent and opaque objects (Xu et al., 2021; Wang et al., 2023), estimating object poses, and monitoring the state of materials synthesis, such as detecting if a solution is dissolved (Shiri et al., 2021). Dexterous manipulation and handling are also necessary, such as constrained motion for picking and transporting objects without spilling their contents, pouring skills, and manipulation of tools and objects. Additionally, high precision and repeatability are crucial for reproducible and reliable results in robot-executed chemistry experiments.

Ensuring safety during experiments and interactions is another challenge (Ménard & Trant, 2020). Multi-layered safety requirements are necessary, including high-level constraints on material synthesis order in experiment description and task planning, and low-level manipulation and perception skills to prevent spilling during transportation of chemistry vials and beakers.

Contributions We introduce an autonomous robotic system for chemistry lab automation, an end-to-end closed-loop robotic framework that translates natural language into structured long-horizon task plans and then executes them by a constrained task and motion planning solver, integrated with perception and manipulation skills, and vision-based evaluation of experiment outcomes (Fig. 1). Our approach consists of two modules. The first is CLAIRIFY, which translates a natural language input into a structured plan. The second is the translation of the plan to low-level robot actions. To evaluate the framework, we use a domain-specific language called Chemical Description Language (XDL) (Mehr et al., 2020). XDL is an XML-based domain-specific language used to describe action plans for chemistry experiments in a structured format, and is suitable for commanding robots in self-driving laboratories (Seifrid et al., 2022). XDL is hardware-agnostic, meaning that it can be executed on many robot platforms and lab setups. We showcase an example of how XDL can be executed on a Franka robot in a chemistry lab setting by converting XDL to low-level robot actions using TAMP. Our method includes (I) a rule-based iterative verifier to check for syntax correctness and environment constraints, which improves zero-shot task plan generation in a data-scarce domain. (II) At the TAMP level, constrained task and motion planning is incorporated via a PDDLStream (Garrett et al., 2020) solver to avoid spillage when transporting liquids and powders. When adding constraints to the robot's degrees of freedom, the motion planning success rate was lower compared to unconstrained planning. To overcome this difficulty, we introduced a new degree of freedom to our robot. We demonstrate that an 8-DoF robot has 97% success rate in constrained TAMP compared to 84% of a 7-DoF robot. Moreover, we present accurate and efficient pouring skills inspired by human motions with an average relative error of 8.1% and 8.8% for pouring water and salt, respectively, compared to a baseline method with 81.4% and 24.1% errors. Our method achieves comparable results to recent studies (Kennedy et al., 2019; Huang et al., 2021), yet it stands out for its simplicity and the reduced need for complex sensor feedback.

Our evaluation results demonstrate that CLAIRIFY outperforms the current state-of-the-art model for XDL generation presented in (Mehr et al., 2020). Additionally, our framework represents an advancement from the approach in (Fakhruldeen et al., 2022), which relied on a finite state machine with fixed objects in a static workspace. Our framework perceives the environment and plans long-horizon trajectories to perform multistep chemistry experiments. It can close the loop at two levels. The first is at the chemistry task level, where online visual feedback is used to estimate the progress of task execution. The second is at the robot motion planning and skill level, where the robot is able to adapt to uncertainties, for example, by refining its plan at execution time using visual feedback. As a proof of concept for chemistry lab automation, we achieved results that are comparable to the literature ground truth for the *solubility* experiment, with a 7.2% error rate for the solubility of salt, and successful recrystallization of alum.

The paper is organized as follows: Section 2 reviews the state of the art. Section 3 defines the problem and presents the proposed end-to-end approach, covering natural language and perceptual inputs to robot task and motion planning and skill execution. Experiments and results are presented in Section 4. Discussion of the results is provided in Section 5, and conclusions are drawn in Section 6.

2 Related work

This section describes recent advancements in lab automation, specifically focusing on robotics and the methods through which large language models (LLMs) can be incorporated into these systems. Furthermore, the section highlights the challenges associated with generating verifiable task plans from LLMs, which are necessary to generate robot tasks and motion plans. Lastly, the section outlines recent efforts that focus on identifying the essential robot skills required to execute lab automation tasks effectively.

2.1 Lab automation

Lab automation aims to introduce automated hardware in a laboratory to improve the efficiency of scientific discovery. It has been applied to high-throughput screening (Pereira & Williams, 2007) (Macarron et al., 2011) to discover desired materials from a pool of candidates. With the rise of AI technologies, the concept of a self-driving laboratory (SDL) (Häse et al., 2019; Seifrid et al., 2022) that combines experiment planning by AI and automated experimentation platforms has emerged. A review on SDL can be found in (Abolhasani & Kumacheva, 2023). Different hardware for lab automation has been utilized in SDLs to meet the needs of individual laboratories, such as pipetting robots (Higgins et al., 2021) or flow reactors (Epps et al., 2020; Li et al., 2020). The Chemputer (Mehr et al., 2020) is an example of a specialized machine for automated chemistry experiments. In the corresponding paper, the authors demonstrated an automated workflow that translates organic chemistry literature into a structured language called XDL and synthesized the specified molecules. While specialized hardware for chemistry is widely used in lab automation, general-purpose robots have also been applied to chemistry because of their flexibility, mobility, and dexterous manipulation capabilities. The utility of mobile general-purpose robots for discovering improved photocatalysts for hydrogen production was demonstrated in (Burger et al., 2020). ARChemist (Fakhruldeen et al., 2022),

a lab automation system, was developed to conduct experiments, including solubility screening and crystallization, without human intervention. Another work (Knobbe et al., 2022) endowed a collaborative robot with a force-sensitive pipetting skill by extending the robot with a pipetting finger system. Differently from other works where pick, place, insertion, or liquid handling, were the focus of robot skills, in (Pizzuto et al., 2022), the robot learns to scrape powders from vials for crystallization experiments. During the experiment, crystallized materials tend to adhere to the inner wall of the vial. Acquiring these crystals necessitates physical interaction between the crystals and the vial using a robot arm, which is unattainable with a valve-pump-based system. These examples demonstrate the potential utilization of a versatile robot in expanding the scope of tasks automated within chemistry laboratories. Similarly, an automated chemistry experiment system that mimics the motions of human chemists has been proposed in (Lim et al., 2020). Additionally, The properties of thin-film materials were optimized using a robot arm in (MacLeod et al., 2020). Although these major steps towards chemistry lab automation have been made, their dependence on predefined tasks and on motion plans without constraint satisfaction guarantees limits their flexibility in new and dynamic workspaces. In those works, pick & place was the primary task that the manipulators were carrying out. Those works were tested in hand-tuned and static environments to avoid occurrences of unsatisfied task constraints and the associated problems, such as chemical spills during the transfer of vessels filled with liquid. Our framework resolves these gaps through using large language models to generate long-horizon machine-readable instructions and passing them to a constraint satisfaction and scene-aware planning system with a variety of skills.

2.2 Large language models for chemistry

Several language models specialized for the chemistry or science domain have been proposed, such as MolT5 (Edwards et al., 2022), Chemformer (Irwin et al., 2022), and Galactica (Taylor et al., 2022). After the release of GPT-3, chemistry applications were attempted without further training (Jablonka et al., 2023). The abilities to do Bayesian optimization (Ramos et al., 2023), to use external chemistry tools (Bran et al., 2023), and to synthesize molecules by reading documentation (Boiko et al., 2023) were explored. Our work focuses on increasing the reliability of the output of LLMs without further training by introducing iterative prompting and low-level planning through a task and motion planning framework.

2.3 Leveraging language models with external knowledge

A challenge with LLMs generating code is that the correctness of the code is not assured. There have been many interesting works on combining language models with external tools to improve the reliability of the output. Mind's Eye (Liu et al., 2023) attempts to ground large language model's reasoning with physical simulation. They trained LLM with pairs of language and codes and used the simulation results to prompt an LLM to answer general reasoning questions.

Toolformer (Schick et al., 2023) incorporates API calls into the language model to improve a downstream task, such as question answering, by fine-tuning the model to learn how to call the API. LEVER (Ni et al., 2023) improves LLM prompting for SQL generation by using a model-based verifier trained to verify the generated programs. As SQL is a common language, the language model is expected to understand its grammar. However, for domain-specific languages, it is difficult to acquire training datasets and expensive to execute the plans to verify their correctness. Our method does not require fine-tuning any models. Furthermore, there is no need for prior knowledge of the target language within the language model obtained during the training phase. Our idea is perhaps closest to LLM-AUGMENTER (Peng et al., 2023), which improves LLM outputs by giving it access to external knowledge and automatically revising prompts in natural language question-answering tasks. Our method similarly encodes external knowledge in the structure of the verifier and prompts, but for a structured and formally verifiable domain-specific language. A review on augmenting LLMs with external tools is found in (Mialon et al., 2023).

2.4 Task planning with large language models

High-level task plans are often generated from a limited set of actions (Garrett et al., 2020), because task planning becomes intractable as the number of actions and time horizon grows (Kaelbling & Lozano-Pérez, 2011). One approach to do task planning is using rule-based methods (Mehr et al., 2020; Baier et al., 2009). More recently, it has been shown that models can learn task plans from input task specifications (Sharma et al., 2021; Mirchandani et al., 2021; Shah et al., 2021), for example using hierarchical learning (Xu et al., 2018; Huang et al., 2019), regression based planning (Xu et al., 2019), or reinforcement learning (Eysenbach et al., 2019). However, to effectively plan tasks using learning-based techniques, large datasets are required that are hard to collect in many real-world domains.

Recently, many works have used LLMs to translate natural language prompts to robot task plans (Ahn et al., 2022; Huang et al., 2022; Liang et al., 2022; Singh et al., 2022). For

example, Inner Monologue (Huang et al., 2022) uses LLMs in conjunction with environment feedback from various perception models and state monitoring. However, because the system has no constraints, it can propose plans that are nonsensical. SayCan (Ahn et al., 2022), on the other hand, grounds task plans generated by LLMs in the real world by providing a set of low-level skills the robot can choose from. A natural way of generating task plans is using code-writing LLMs because they are not open-ended (i.e., they have to generate code in a specific manner in order for it to be executable) and are able to generate policy logic. Several LLMs trained on public code are available, such as Codex (Chen et al., 2021), CodeT5 (Wang et al., 2021), AlphaCode (Li et al., 2022) and CodeRL (Le et al., 2022). LLMs can be prompted in a zero-shot way to generate task plans. For example, Huang et al. (2022) analyzed the planning ability of LLM in virtual environment, Code as Policies (Liang et al., 2022) repurposes code-writing LLMs to write robot policy code, and ProgPrompt (Singh et al., 2022) generates plans that take into account the robot's current state and the task objectives. PaLM-E is an embodied LLM that translates visual, state estimates, sensory data, and language domains into embodied reasoning for robot planning (Driess et al., 2023). Text2Motion (Lin et al., 2023) combines LLM with skill feasibility heuristics to guide task planning. LLM-GROP (Ding et al., 2023) demonstrated human-aligned object rearrangement from natural-language commands combined with TAMP. Inagaki et al. (2023) generated Python code for an automated liquid-handling robot from natural language instructions. However, these methods generate Pythonic code, which is abundant on the Internet. For domain-specific languages, naive zero-shot prompting is not enough; the prompt has to incorporate information about the target language so that the LLM can produce outputs according to its rules.

Our approach, on the other hand, generates a task plan directly from an LLM in a zero-shot way on a constrained set of tasks that are directly translatable to robot actions. We ensure that the plan is syntactically valid and meets environment constraints using iterative error checking. However, while the generated plan is verified for syntax and constraint satisfaction, it does not consider the robot embodiment and workspace scene, making its execution on a robot uncertified. To address this issue, we integrate the generated task plans as intermediate goals into a certifiable task and motion planner framework, which produces executable trajectories for the robot.

2.5 Task and motion planning with constraints

Task and motion planning (TAMP) (Garrett et al., 2021) simultaneously determines the sequence of high-level symbolic actions, such as picking and placing, and low-level motions for the action, such as trajectory generation. Different approaches have been proposed in the literature to solve the TAMP problem. For example, (Toussaint, 2015) proposed a non-linear constrained programming formulation for TAMP problems. In another work from the same group, (Toussaint et al., 2018) integrated TAMP with kinematic or dynamic constraints for tool-use and manipulation. Another TAMP solver, PDDLStream (Garrett et al., 2020), extends PDDL (Ghallab et al., 1998), a common language to describe a planning problem mainly targeting discrete actions and states, by introducing streams, a declarative procedure via sampling procedures. PDDLStream reduces a continuous problem to a finite PDDL problem and invokes a classical PDDL solver as a subroutine. Although pure-planning approaches to TAMP is general, it is computationally inefficient. To accelerate the planning, an approach to incorporate geometric information has been proposed (Dantam et al., 2018). Recently, geometric information also has been used with learning-based approaches to improve planning efficiency (Khodeir et al., 2023; Kim et al., 2022). In (Driess et al., 2020), an initial scene image is inputted to a neural network that predicts the robot's discrete action, and subsequently, a motion planning problem is solved.

Another important aspect to take into account when solving a TAMP problem is the incorporation of safety measures and constraints. Since PDDLStream verifies the feasibility of action execution during planning time, it can inherently enhance safety by avoiding unfeasible plans or plans that may lead to unsafe situations. Nonetheless, PDDLStream does not yet account for constraints in the planning process, for example, to avoid material spillage from beakers during transportation, which impedes its deployment in real-world lab environments. For this purpose, sampling-based motion constraint adherence (Berenson et al., 2011) or model-based motion planning (Muchacho et al., 2022) are possible stream choices. To overcome this shortcoming, our work extends PDDLStream with a projection-based sampling technique (Kingston et al., 2019) to provide constraint satisfaction, completeness, and global optimality. The proposed PDDLStream takes intermediate goals generated by LLMs in a structured language as its input.

2.6 Skills and integration of chemistry lab tools

In the process of lab automation, robots interact with tools and objects within the workspace and require a repertoire of many laboratory skills. Some skills can be completed with existing heterogeneous instruments and sensors in chemistry labs, such as scales, stir plates, pH sensors, and heating instruments. Other skills are currently done either manually by humans in the lab or with expensive special instruments. In a self-driving lab, robots should acquire those skills by effectively using different sensory inputs to compute

appropriate robot commands. Pouring is a common skill in chemistry labs. Recent work (Kennedy et al., 2019; Huang et al., 2021) used vision and weight feedback to pour liquid with manipulators. (Kennedy et al., 2019) proposed using optimal trajectory generation combined with system identification and model priors. To achieve milliliter accuracy in water pouring tasks with a variety of vessels at human-like speeds, (Huang et al., 2021) used self-supervised learning from human demonstrations. In this work, we have reached similar results for pouring, using commercial scales that have delayed feedback. Our approach is model-free, and it can pour granular solids as well. Granular solids have different dynamics from liquids, similar to the avalanche phenomenon. Lastly, while executing a chemistry experiment, the robot should possess perception skills to measure progress toward completing the task. For example, in solubility experiments, the robot should perceive when the solution is fully dissolved, and therefore stop pouring the solvent into the solution. There are different ways to measure solubility. In our work, we use the turbidity measure (Shiri et al., 2021), which is based on optical properties of light scattering and absorption by suspended sediment (Kitchener et al., 2017).

3 Methods

We propose an automated robotic experiment platform that takes instructions from a human in natural language and executes the corresponding experiment. The natural language input is converted into a sequence of robot plans written in a structured language by an LLM-based system, CLAIRIFY. XDL (Steiner et al., 2019) was used as the robot programming language. The task and motion planning module generates the robot motion from the generated XDL. The overview of the proposed method is shown in Fig. 2.

3.1 CLAIRIFY: natural language to structured programs

CLAIRIFY takes a chemistry experiment description in natural language and generates a structured experiment plan in XDL format, which will be fed into the subsequent module to generate robot motions. A general overview of the CLAIRIFY pipeline is given in Fig. 2a.

CLAIRIFY generates XDL with an automated iterative prompting between a *generator* and a *verifier*. The generator outputs XDL from a prompt that combines the experiment description and the target language format description. However, we cannot guarantee the output from the generator is syntactically valid, meaning that it would definitely fail to compile into lower-level robot actions. To generate syntactically valid programs, we pass the output of the generator through a verifier. The verifier determines whether the generator output follows all the rules and specifications of the target structured language and can be compiled without errors. If it cannot, the verifier returns error messages stating where the errors were found and what they were. These are then appended to the generator output and added to the prompt for the next iteration. This process is repeated until a valid program is obtained, or until the timeout condition is reached. Algorithm 1 describes our proposed method.

Once the generator output passes through the verifier with no errors, we are guaranteed that it is a syntactically valid structured language. This output will then be translated into lower-level robot actions by passing it through TAMP for robot execution. Each component of the pipeline is described in more detail below.

3.1.1 Generator

The generator takes a user's instruction and generates unverified structured language using an LLM. The input prompt to the LLM is composed of a description of the target language, a sentence specifying what the LLM should do (i.e. "Convert to XDL"), the command to the LLM, and the natural language instruction for which the task plan should be created. The description of the XDL language includes its file structure and lists of the available actions (which can be thought of as functions), their allowed parameters, and their documentation. The input prompt skeleton is shown in Snippet 1, Fig. 3.

Although the description of the target structured language is provided, the output may contain syntactic errors. To ensure syntactical correctness, the generator is iteratively prompted by the automated interaction with the verifier. The generated code is passed through the verifier, and if no errors are generated, then the code is syntactically correct. If errors are generated, we re-prompt the LLM with the incorrect task plan from the previous iteration along with the list of errors indicating why the generated steps were incorrect. The skeleton of the iterative prompt is shown in Snippet 2, Fig. 3. The feedback from the verifier is used by the LLM to correct the errors from the previous iteration. This process is continued until the generated code is error-free or a timeout condition is reached, in which case the system reports not being able to generate a task plan.

3.1.2 Verifier

The verifier works as a syntax checker and static analyzer to check the output of the generator and send feedback to the generator. It first checks whether the input can be parsed as correct XML and then checks the allowance of action tags, the existence of mandatory properties, and the correctness of optional properties using a rule-based method that checks for permissible functions and parameters in the XDL docu-

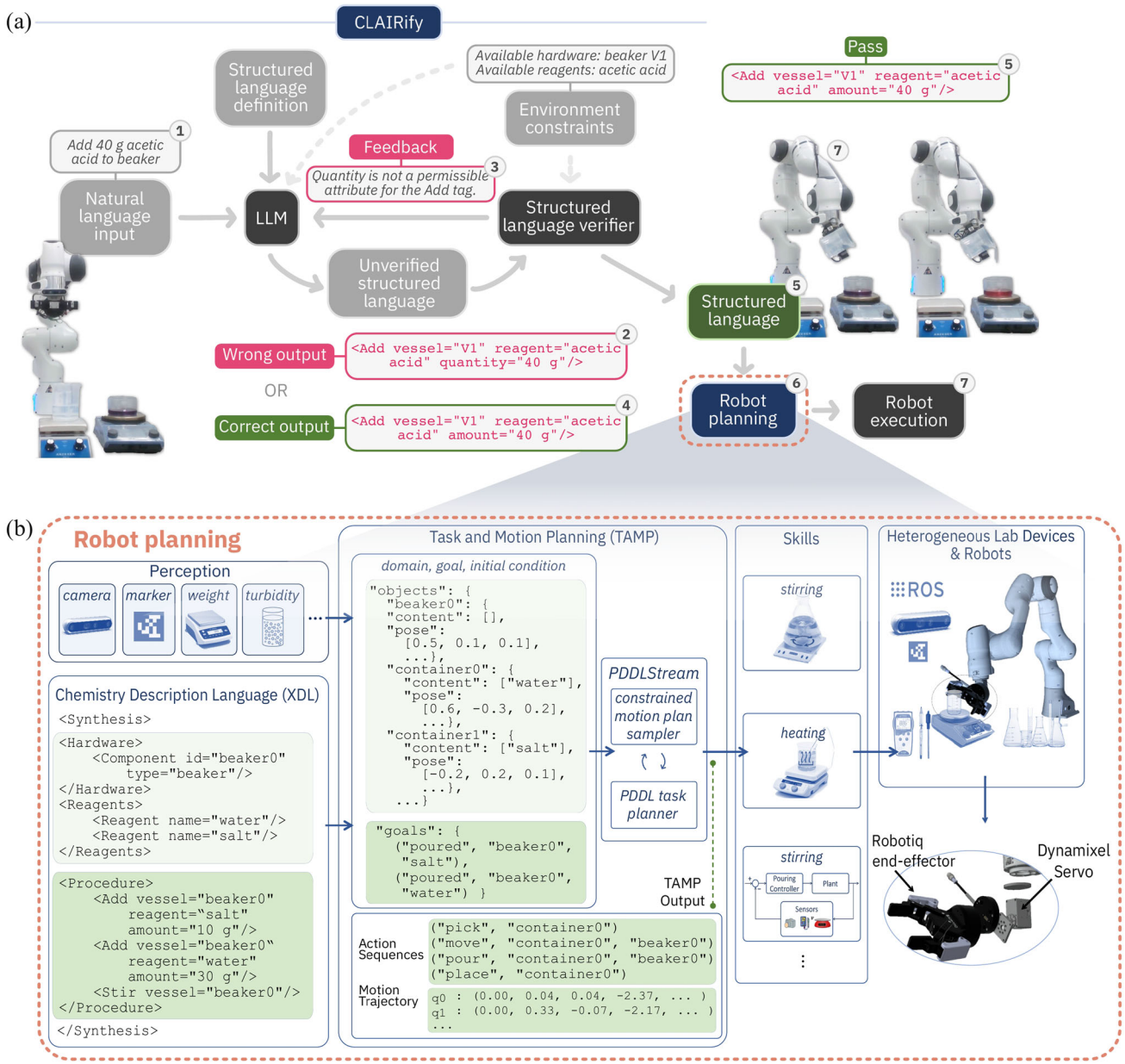


Fig. 2 Our framework. **a** CLAIRIFY: LLM-based natural language processing module. The LLM takes the input (1), structured language definition, and (optionally) resource constraints and generates unverified structured language (2). The output is examined by the verifier and is passed to LLM with feedback (3). The LLM-generated outputs pass through the verifier (4). The correct output (5) is passed to the task and motion planning module (6) to generate robot trajectories. **b** Robot planning module, which is composed of *Perception*, *Task and Motion Planning*, and *Skills* blocks. Our framework enables the robot to leverage available chemistry lab devices (including sensors and actuators) by adding them to the robot network through ROS. The robot is equipped with an additional DoF at the end-effector, allowing it to perform constrained motions. Our framework receives the chemical synthesis goal in XDL format. The *procedure* component is converted into corresponding PDDL goals, and *hardware* and *reagents* components identify the required initial condition for synthesis. *Perception* detects objects and estimates their positions, contents in the workspace, and task progress. PDDLStream generates a sequence of actions for the robot execution (7)

Algorithm 1 CLAIRIFY: Verifier-Assisted Iterative Prompts

Input: Structured language description $\mathcal{L}$, instruction x
Output: Structured language task plan, y_{SL}
  procedure ITERATIVEPROMPTING($\mathcal{L}$, x)
    $y_{SL'} = \text{Generator}(\mathcal{L}, x)$
    errors $= \text{Verifier}(y_{SL'})$
    while len(errors) > 0 and timeout condition != True **do**
      $y_{SL'} = \text{Generator}(\mathcal{L}, x, y_{SL'}, \text{errors})$
      errors $= \text{Verifier}(y_{SL'})$
    end while
    $y_{SL} = y_{SL'}$
  end procedure

```
initial_prompt = """
# <Description of XDL>

# <Hardware constraints(optional)>
# <Reagent constraints (optional)>

Convert to XDL:
# <Natural language instruction>
"""
```

Snippet 1: Initial prompt

```
iterative_prompt = """
# <Description of XDL>

# <Hardware constraints(optional)>
# <Reagent constraints (optional)>

Convert to XDL:
# <Natural language instruction>
# <XDL from previous iteration>
This XDL was not correct.
There were the errors
# <List of errors, one per line>
Please fix the errors
"""
```

Snippet 2: Iterative prompt

Fig. 3 Prompt skeleton: (1) At the initial generation, we prompt the LLM with a description of XDL and the natural language instruction. (2) After the LLM generates structured-language-like output, we pass it through our verifier. If there are errors in the generated program, we concatenate the initial prompt with the XDL from the previous iteration and a list of the errors. The full prompt can be viewed in "Appendix B"

mentation. This evaluates if the input is syntactically correct XDL. The verifier also checks the existence of definitions of hardware and reagents used in the procedure or provided as environment constraints, which works as a basic static analysis of necessary conditions for executability. If the verifier catches any errors, the candidate task plan is considered to be invalid. In this case, the verifier returns a list of errors it found, which is then fed back to the generator. The role of the verifier is limited to pointing out the errors, and it does not propose how to fix them. To propose a correct modification, the verifier requires an understanding of the meaning of the input. However, semantic understanding is beyond the ability of a rule-based system. Therefore, in CLAIRIFY, the LLM-based generator fixes the errors using the feedback messages

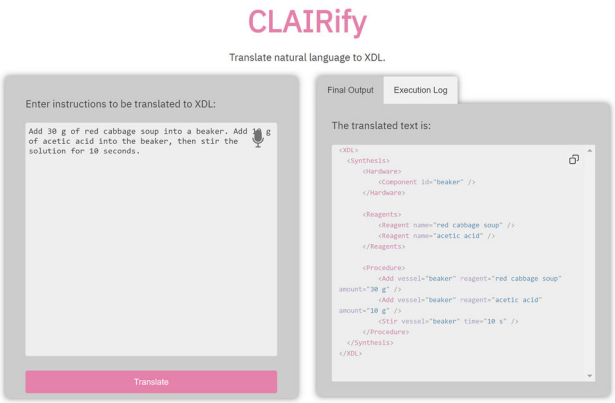


Fig. 4 Web interface for CLAIRIFY. Users input natural language descriptions of the experiment in the left column. XDL is generated in the right column when the user pushes the Translate button

from the verifier. The error message is designed to be concise to save the context length of LLM.

3.1.3 Incorporating environment constraints

Because resources in a robot workspace are limited, we need to consider those constraints when generating task plans. If specified, we include the available resources in the generator prompt. The verifier also catches if the candidate plan uses any resources aside from those mentioned among the available robot resources. Those errors are included in the generator prompt for the next iteration. If a constraint list is not provided, we assume the robot has access to all resources. In the case of chemistry lab automation, those resources include experiment hardware and reagents.

3.1.4 User interface

We provide a graphical user interface for CLAIRIFY to increase accessibility. Users can access it via a web browser and CLAIRIFY is called by the Python backend implemented in Flask (Grinberg, 2018). In Fig. 4, we show the user interface. The user enters an experiment (in the figure, we show Experiment 0 from the Chem-EDU dataset). After pressing the Translate button, the interface shows the execution log and generated XDL in the right panel with syntax highlighting. The time taken to generate XDL from a natural language using CLAIRIFY is mainly dependent on the OpenAI server response time and the number of times the generator is called. For the experiment in Fig. 4, we measured the translation time. These measurements were performed on three separate occasions, spanning different days and times. On average, the generation process took approximately 33 ± 3 s per iteration and required two generator calls.

Algorithm 2 TAMPFORLABAUTOMATION()

Input: A XDL recipe χ, sensory input $\mathcal{H}$, PDDLStream domain $\mathcal{D}$
Output: Reference *plan* to execute
1: $Goals, \mathcal{O} \leftarrow$XDLPARSER($\chi$) ▷ Objects
2: $\mathcal{I} \leftarrow$ PERCEPTION($\mathcal{H}, \mathcal{O}$) ▷ Initial conditions
3: **if** not PASSCONDITIONS($\mathcal{I}$, $\mathcal{O}$) **then return**
4: $plan = \emptyset, \mathcal{P} = \emptyset$
5: **for all** $\mathcal{G} \in Goals$ **do**
6: **while** $time() \leq t_{max}$ **do**
7: $\mathcal{P}$ = OPTIMISTICPDDLSTREAMPLAN($\mathcal{I}$, $\mathcal{G}$, $\mathcal{D}$)
8: **if** $\mathcal{P} \neq \emptyset$ and ISSTREAMFEASIBLE($\mathcal{P}$) **then break**
9: **end while**
10: **if** $\mathcal{P} = \emptyset$ **then return**
11: $plan \leftarrow plan \cup \mathcal{P}$
12: $\mathcal{I}$ = UPDATESCENEREPRESENTATION($\mathcal{I}, \mathcal{P}$)
13: **end for**
14: **return** *plan*

3.2 Task and motion planning for chemistry experiments

Our task plan execution framework consists of three components: *perception*, *task and motion planning* (TAMP), and a set of manipulation *skills*, as shown in Fig. 2b. XDL input coming from CLAIRIFY provides a high-level description of experiment instructions to the TAMP module. The perception module updates the scene description by detecting the objects and estimating their positions using fiducial markers. We used AprilTag (Olson, 2011). Currently, we assume prior knowledge of vessel contents and sizes, and each vessel is mapped to a unique marker ID. Given the instructions from XDL and the instantiated workspace state information from perception, a sequence of high-level actions and robot trajectories are simultaneously generated by our PDDLStream TAMP solver (Yoshikawa et al., 2023). The resulting plan is then realized by the manipulation module and robot controller, while closing the loop with perception feedback, such as updated object positions and status of the solution.

The TAMP module converts experiment instructions given by XDL into PDDLStream goals and generates a motion plan. The TAMP algorithm is shown in Algorithm 2.

3.2.1 PDDLStream

A PDDLStream problem described by a tuple ($\mathcal{P}$, $\mathcal{A}$, $\mathcal{S}$, $\mathcal{O}$, $\mathcal{I}$, $\mathcal{G}$) is defined by a set of predicates $\mathcal{P}$, actions $\mathcal{A}$, streams $\mathcal{S}$, initial objects $\mathcal{O}$, an initial state $\mathcal{I}$, and a goal state $\mathcal{G}$. A predicate is a boolean function that describes the logical relationship of objects. A logical action $a \in \mathcal{A}$ has a set of preconditions and effects. The action a can be executed when all the preconditions are satisfied. After execution, the current state changes according to the effects. The set of streams, $\mathcal{S}$, distinguishes a PDDLStream problem from traditional PDDL. Streams are conditional samplers that yield objects that satisfy specific constraints. The goal of PDDLStream planning is to find a sequence of logical actions and a continuous motion trajectory starting from the initial state until all goals are satisfied, ensuring that the returned plan is valid and executable by the robot. We define four types of actions in our PDDLStream domain: *pick*, *move*, *place*, and *pour*. For example, the *move* action translates the robot end-effector from a grasping pose to a placing or pouring pose using constrained motion planning. PDDLStream handles continuous motion using streams. Streams generate objects from continuous variables that satisfy specified conditions, such as feasible grasping pose and collision-free motion. An instance of a stream has a set of certified predicates that expands $\mathcal{I}$ and functions as preconditions for other actions.

A PDDLStream problem is solved by invoking a classical PDDL planner, such as Fast Downward (Helmert, 2006), with optimistic instantiation of streams (line 7, Algorithm 2). If a plan for the PDDL problem is found, the optimistic stream instances $s \in \mathcal{S}$ in the plan are evaluated to determine the actual feasibility (line 8). If no plan was found or the streams are not feasible, other plans are explored with a larger set of optimistic stream instances.

Chemical description language (XDL) XDL is based on XML syntax and is mainly composed of three mandatory sections: Hardware, Reagents, and Procedure. We parse XDL instructions and pass them to the TAMP module. The Hardware and Reagents sections are parsed as initial objects $\mathcal{O}$. Procedure is translated into a set of goals $Goals$ (line 1, Algorithm 2). $\mathcal{I}$ is generated from $\mathcal{O}$ and sensory inputs (line 2). Each intermediate goal $\mathcal{G} \in Goals$ is processed by PDDLStream (line 5). If a plan to attain $\mathcal{G}$ is found, it is stored (line 10) and $\mathcal{I}$ is updated according to the plan (line 11). After a set of plans to attain all goals is found, we obtain a complete motion plan (line 12).

Plan refinement at execution time We adopt two considerations for the dynamic nature of chemistry experiments: motion plan refinement and task plan refinement.

The generated motion plan is refined to reflect the updated status of the scene and to overcome perception errors. The initial object pose detection may contain errors, therefore, the object may not be present in the expected position during execution. This error arises for two reasons. First, when the robot interacts with the objects in the workspace, their position changes, for example when regrasping an object after placing it in the workspace. This change is not always foreseeable by the planner ahead of time. Second, the perception error is lower when the grasping pose is estimated when the robot in-hand camera is closer to the target object, considering the hand-eye calibration error. Lowering the perception error makes the execution more robust to grasping failures. Therefore, to improve the success rate, the object pose is estimated just before grasping, and the trajectory is refined. We assume that the perturbation of the perceived state of the objects is bounded so that it does not cause a change

in the logical state of the system, which would necessitate task-level replanning.

In addition to motion refinement, we consider task plan refinement. Task execution can be repeated using the feedback from perception modules at execution time to support conditional operations in chemistry experiments, such as adding acid until pH reaches 7. The number of repetitions required to satisfy conditions is unknown at planning time, so the task plan is refined at execution time.

3.2.2 Motion constraints for spillage prevention

Unlike pick-and-place of solid objects, robots in a chemistry lab need to carry beakers that contain liquids, powders, or granular materials. These chemicals are sometimes harmful, so the robot motion planner should incorporate constraints to prevent spillage. To this end, an important requirement for robot motion is the orientation constraints of the end-effector. To avoid spillage, the end-effector orientation should be kept in a limited range while beakers are grasped. We incorporated constrained motion planning in the framework to meet these safety requirements, under the assumption of velocity and acceleration upper bounds. Moreover, we introduced an additional (8th) degree of freedom to the robot arm, in order to increase the success rate of constrained motion planning. We empirically observed no spillage as long as orientation constraints are satisfied in the regular acceleration and velocity of the robot end-effector, particularly since beakers are typically not filled to their full capacity in a chemistry lab.

Algorithm 3 CONSTRAINEDMOTIONPLANNING()

```
1: for all i ∈ trials do
2:     q_g ← SOLVEIK((^I p_B, ^I R_B))
3:     PATHPLANNER ← init(q_0, q_g)
4:     while path is ∅ do
5:         q ← SAMPLE()
6:         while ‖F(q)‖ > ϵ do
7:             δq ← J^†(q)F(q)
8:             q ← q − δq
9:         end while
10:        path ← PATHPLANNER(q)
11:    end while
12:    if path ≠ ∅ then return path
13: end for
14: return path
```

Constrained motion planning Given a robot with n degrees of freedom in the workspace $\mathcal{Q} \in \mathbb{R}^n$ with obstacle regions $\mathcal{Q}_{obs} \in \mathbb{R}^n$, the constrained planning problem can be described as finding a path in the manipulator's free configuration space $\mathcal{Q}_{free} = \mathcal{Q} - \mathcal{Q}_{obs}$ that satisfies initial configuration $\boldsymbol{q}_0 \in \mathbb{R}^n$, end-effector goal pose $({}^{\mathcal{I}}\boldsymbol{p}_{\mathcal{B}} \in \mathbb{R}^3, {}^{\mathcal{I}}\boldsymbol{R}_{\mathcal{B}} \in SO(3))$, and equality path constraints $\boldsymbol{\mathcal{F}}(\boldsymbol{q}) : \mathcal{Q} \to \mathbb{R}^k$. The constrained configuration space can be represented by the implicit manifold $\mathcal{M} = \{q \in \mathcal{Q} \mid \boldsymbol{\mathcal{F}}(\boldsymbol{q}) = \boldsymbol{0}\}$. The implicit nature of the manifold prevents planners from directly sampling since the distribution of valid states is unknown. Further, since the constraint manifold resides in a lower dimension than the configuration space, sampling valid states in the configuration space is highly improbable and thus impractical. Following the constrained motion planning framework developed in (Kingston et al., 2019, 2018), our framework integrates the projection-based method for finding constraint-satisfying configurations during *sampling* as described in Algorithm 3. In this work, the constraints are set to the robot end-effector, hence they can be described with geometric forward kinematics, with its *Jacobian* defined as $\boldsymbol{\mathcal{J}}(\boldsymbol{q}) = \frac{\delta \boldsymbol{\mathcal{F}}}{\delta \boldsymbol{q}}$. After sampling from $\mathcal{Q}_{free}$ in line 5, projected configurations $\boldsymbol{q}$ are found by minimizing $\boldsymbol{\mathcal{F}}(\boldsymbol{q})$ iteratively using Newton's method (highlighted in grey). We use probabilistic roadmap methods (PRM*) to plan efficiently in the 8-DoF configuration space found in our chemistry laboratory domain (Karaman & Frazzoli, 2011; Kavraki et al., 1996).

The constrained path planning problem is sensitive to the start and end states of the requested path, since paths between joint states may not be possible under strict or multiple constraints. If constrained planning is executed with any arbitrary valid solution from the IK solver, the planner typically fails. To address this shortcoming, three considerations are made. First, a multi-threaded IK solver with both iterative and random-based techniques is executed, and the solution that minimizes an objective function ϕ is returned with TRAC-IK, proposed in (Beeson & Ames, 2015). During grasping and placing, precision is paramount, and we only seek to minimize the sum-of-squares error between the start and goal Cartesian poses. Second, depending on the robot task, the objective function is extended to maximize the manipulability ellipsoid as described in (Yoshikawa, 1985), which is applied for more complicated maneuvers, such as transferring liquids across the workspace. Finally, note that configuration sampling must account for the fact that multiple goal configurations are possible. For this purpose, Algorithm 3 can iterate several times to find various goal configurations in line 2.

8-DoF robot arm To increase the success rate of planning and grasping under non-spillage constraints, we introduced an additional degree of freedom to the 7-DoF Franka robot and mounted the end-effector parallel to the floor as shown in the bottom right of Fig. 2b. The parallel grasping and introduction of a revolute joint facilitate the manipulation of liquids within beakers. Parallel grasping, where the gripper is horizontal, is advantageous over top-down grasping to pour liquids because the robot hand does not touch the rim of a beaker and does not block the flow of liquid. Frequently, when planning constrained motion, the initial 7-DoF robot arm would encounter joint limit issues with parallel grasping poses, rendering the subsequent pouring action impractical.

It is possible to achieve parallel grasping by adjusting the angle of the end-effector without adding an extra revolute joint by connecting the end-effector with a rigid link to the robot. However, the integration of a servo motor directly before the gripper facilitates pouring control, as it enables the accomplishment of pouring actions using a single motor. Limiting the pouring motion to the last joint is also advantageous for the safety of humans near the robot because pouring motion sometimes entails quick rotations. Mounting the robot base on the wall instead of the default tabletop placement is another possible solution to parallel grasping, but it is not supported by the manufacturer. In addition to the advantage in pouring, the 8-DoF robot has a higher empirical success rate in constrained motion planning in the parallel grasping pose, as shown in Sect. 4.4.2, which leads to a higher success rate in total task and motion planning. As a result, adding one degree of freedom was the most suitable solution for our purpose.

3.2.3 Manipulation and perception skills

Chemistry lab skills require a particular suite of sensors, algorithms, and hardware. We provide an interface for instantiating different skill instances through ROS and simultaneously commanding them. For instance, recrystallization experiments in chemistry require pouring, heating, and stirring, which uses both weight feedback for volume estimation and skills for interacting with the liquid using available hardware.

Pouring controller In chemistry labs, a frequently used skill in chemical experiments is pouring. Pouring involves high intra-class variations depending on the underlying objective (e.g., reaching a desired weight or pH value); the substances and material types being handled (e.g., granular solids or liquids); the glassware being used (e.g., beakers and vials); the overall required precision; and the availability of accurate and fast feedback. Pouring is a closed-loop process, in which feedback should be continuously monitored. Among these pouring actions, in our work, we consider the following variations: pouring of liquids and pouring of granular solids. Note that, in contrast to many control problems, pouring is a non-reversible process where we cannot compensate for overshoot (as the poured material cannot go back to the pouring beaker if mixed with another material).

Inspired by observations of chemists pouring reagents, we propose a controller that allows the robot to perform different pouring actions. As shown in Fig. 5, the proposed method takes sensor measurements (e.g., weight feedback from the scale) as feedback and a reference pouring target. The algorithm outputs a robot end-effector joint velocity describing oscillations of the arm's wrist. Since sensors are characterized by measurement delays, chemical reactions require time

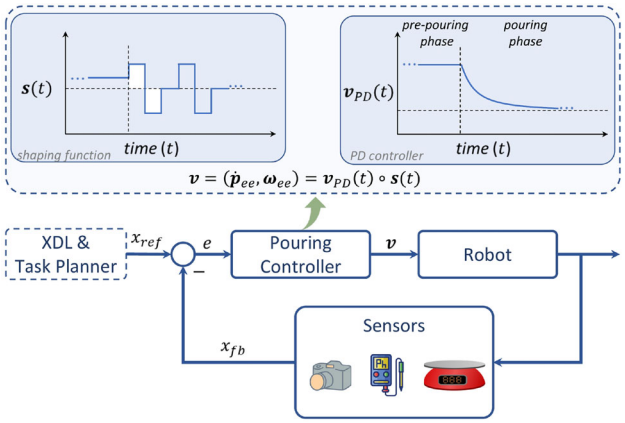


Fig. 5 Pouring skill controller: given the XDL and TAMP reference values and sensor feedback, the pouring controller computes the end-effector velocity for the robot by blending a shaping function $s(t)$ and a PD control output $v_{PD}(t)$

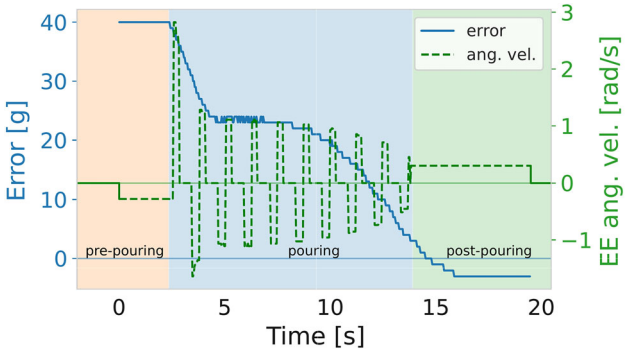


Fig. 6 An example of pouring control. The velocity of the end-effector is controlled based on the feedback error and shaping function

to stabilize, and pouring is a non-reversible action, chemists tend to conservatively pour a small amount of content from the pouring vessel into the target vessel. They periodically wait for some time to observe any effects and then pour micro-amounts again. In our approach, we use a shaping function $s(t)$ to guide the direction and frequency of this oscillatory pouring behavior, while a PD controller lowers the pouring error. The end-effector velocity vector is computed by blending the shaping function $s(t)$ over the PD control signal, $\boldsymbol{v}_{\mathrm{PD}}(t) = \boldsymbol{k}_p e + \boldsymbol{k}_d \dot{e}$, where $e(t) = x_{ref} - x_{fb}$. Figure 6 shows an example of the angular velocity of the end-effector and the error during actual pouring. More information about the pouring skill method can be found in "Appendix A".

Turbidity-based solubility measurement The solubility of a solute is measured by determining the minimum amount of solvent (water) required to dissolve all solutes at a given temperature when the overall system is in equilibrium (Shiri et al., 2021). Since the solutions get transparent when all solutes dissolve into water, turbidity, or opaqueness of the solution, is used as the metric to determine the completion of the exper-

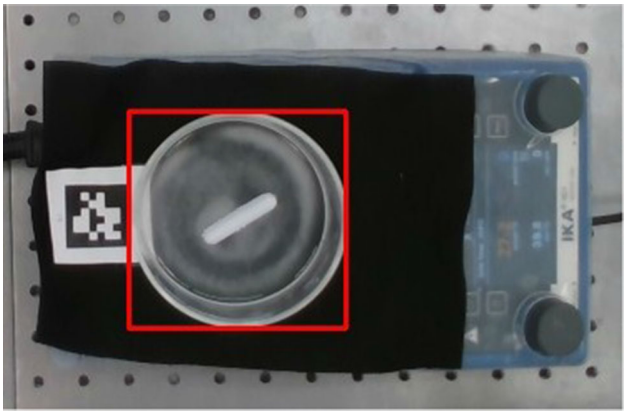


Fig. 7 An example of automated turbidity measurement. The camera detects the Petri dish using Hough Circle Transform. The average brightness of the detected area (red square) is used as a proxy of turbidity (Color figure online)

iment. The average brightness of the solution was used as a proxy for the relative turbidity, inspired by HeinSight (Shiri et al., 2021). That work compared the current measured turbidity value with a reference value (coming from pure solvent) to determine when the solute was dissolved. Differently from them, we use the relative turbidity changes between the current and previous measurement values to detect when the solution is dissolved. Moreover, to make the perception pipeline autonomous, when the robot with an in-hand camera observes the dish containing the solution, it detects the largest circular shape as the dish using a Hough Circle Transform implemented in OpenCV. The square region containing the dish is converted into the HSV color space, and the average Value (brightness) of the region is used as a turbidity value. Figure 7 shows an example of the automated turbidity measurement. Although the detected area contains the dish and stir bar, they do not affect the relative value because these are a constant bias in all measurements.

4 Experiments

4.1 XDL generation

We conducted experiments to evaluate the following hypotheses: i) Automated iterative prompting increases the success rate of unfamiliar language generation, ii) The quality of generated task plans is better than existing methods. To generate XDL plans, we use `text-davinci-003`, the most capable GPT−3.5 model accessible using the OpenAI API at the time of performing experiments (February 2023). We chose to use this instead of `code-davinci-002` due to query and token limits. Additionally, ChatGPT was not yet available through the API at that time.

4.1.1 Datasets

We evaluated our method on two different datasets (Table 1):

Chem-RnD (chemistry research and development) This dataset consists of 108 detailed chemistry-protocols for synthesizing different organic compounds in real-world chemistry labs, sourced from the Organic Syntheses dataset (volume 77) (Mehr et al., 2020a). Due to GPT-3 token limits, we only use experiments with less than 1000 characters. We use Chem-RnD as a proof-of-concept that our method can generate task plans for complex chemistry methods. We do not aim to execute the plans in the real world, and so we do not include any constraints.

Chem-EDU (everyday educational chemistry) We evaluate the integration of CLAIRIFY with real-world robots through a dataset of 42 natural language instructions containing only safe (edible) chemicals and that are, in principle, executable by our robot. The dataset consists of basic chemistry experiments involving edible household chemicals, including acid–base reactions and food preparation procedures.[1] We show some data samples in "Appendix C". When generating the XDL, we also included environment constraints based on what equipment our robot had access to (for example, our robot only had access to a mixing container called "beaker").

4.1.2 Metrics and results

The results section is organized based on the four performance metrics that we will consider, namely: Ability to generate structured-language output, Quality of the generated plans, Number of interventions required by the verifier, and Robotic validation capability. We compared the performance of our method with SynthReader, a state-of-the-art XDL generation algorithm which is based on rule-based tagging and grammar parsing of chemical procedures (Mehr et al., 2020).

1. Ability to generate a structured language plan. First, we investigate the success probability for generating plans. For CLAIRIFY, if it is in the iteration loop for more than x steps (here, we use $x = 10$), we say that it is unable to generate a plan and we exit the program. When comparing with SynthReader, we consider that approach unable to generate a structured plan if the SynthReader IDE (called ChemIDE) throws a fatal error when asked to create a plan.[2] For both models, we also consider them unable to generate a plan if the generated plan only consists of empty XDL

[1] CLAIRIFY Data & code: https://github.com/ac-rad/xdl-generation/.

[2] ChemIDE using XDL: https://croningroup.gitlab.io/chemputer/xdlapp/.

Table 1 Comparison of our method with existing methods on the number of successfully generated valid XDL plans and their quality on 108 organic chemistry experiments from (Mehr et al., 2020a) by 10 expert chemists

Dataset	Method	Number generated ↑	Expert preference↑
Chem-RnD	SynthReader (Mehr et al., 2020)	92/108	13/108
	CLAIRIFY [ours]	**105/108**	**75/108**
Chem-EDU	SynthReader (Mehr et al., 2020)	0/42	–
	CLAIRIFY [ours]	**42/42**	–

Results of the best-performing models for a given metric and dataset are given in bold

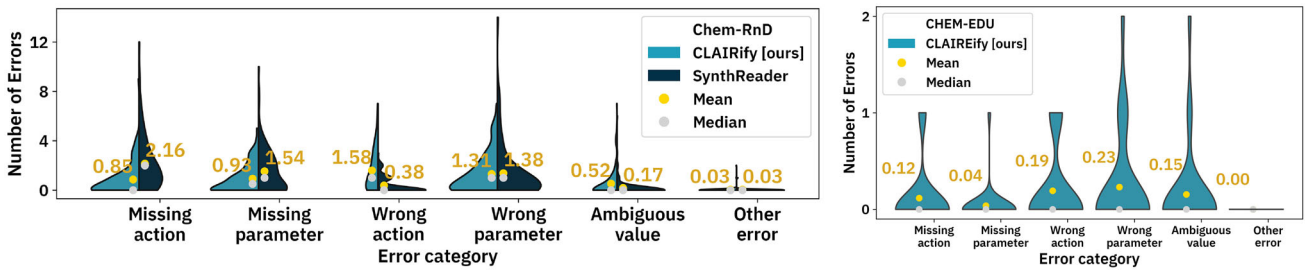


Fig. 8 Violin plots showing distributions of different error categories in XDL plans generated for experiments for the Chem-RnD (left) and Chem-EDU (right) datasets. The x-axis shows the error categories and the y-axis shows the number of errors for that category (lower is better). For the Chem-RnD dataset, we show the error distributions for both CLAIRIFY and SynthReader. Each violin is split in two, with the left half showing the number of errors in plans generated from CLAIRIFY (teal) and the right half showing those from SynthReader (navy). For the Chem-EDU dataset, we only show the distributions for CLAIRIFY. In both plots, we show the mean of the distribution with a gold dot (and the number beside in gold) and the median with a grey dot (Color figure online)

tags (i.e., no experimental protocol). For all experiments, we count the total number of successfully generated language plans divided by the total number of experiments. Using this methodology, we tested the ability of the two models to generate output on both the Chem-RnD and Chem-EDU datasets. The results for both models and both datasets are shown in Table 1. We find that out of 108 Chem-RnD experiments, CLAIRIFY successfully returned a plan 97% of the time, while SynthReader returned a plan 85% of the time. For the Chem-EDU dataset, CLAIRIFY generated a plan for all instructions. SynthReader was unable to generate any plans for that dataset, likely because the procedures are different from typical chemical procedures (they use simple action statements). This demonstrates the generalizability of our method: we can apply it to different language styles and domains and still obtain coherent plans.

2. Quality of the predicted plan (without executing the plan). To determine if the predicted task plans actually accomplish every step of their original instructions, we report the number of actions and parameters that do not align between the original and generated plan, as annotated by expert experimental chemists. To compare the quality of the generated plans between CLAIRIFY and SynthReader, we ask expert experimental chemists to, given two anonymized plans, either pick a preferred plan among them or classify them as equally good. We also ask them to annotate errors in the plans in the following categories: Missing action, Missing parameter, Wrong action, Wrong parameter, Ambiguous value, Other error. Here, actions refer to high-level steps in the procedure (e.g., `<Add reagent="acetic acid">` is an action) and parameters refer to reagents, hardware, quantities and experiment descriptors (e.g., in `<HeatChill vessel ="beaker" temp="100C">`, vessel and temp are both parameters).

The chemists were 10 graduate students and postdoctoral fellows from the University of Toronto and ETH Zürich. For the first 20 experiments, the chemists labelled them together as a group to resolve any labelling ambiguities and then labelled the rest individually (approximately 9 experiments per chemist). The annotations were performed using the LightTag Text Annotation Tool (Perry, 2021).

Chem-RnD dataset The results for the Chem-RnD dataset with respect to expert preference are reported in the last column of Table 1. We found that out of 108 experiments, experts preferred the XDL plan generated from CLAIRIFY 75 times and the one from SynthReader 13 times (the remaining 20 were considered to be of similar quality).

The distributions of the annotated errors are shown in Fig. 8. We find that for 4 out of 6 error categories, our model does at least as well as or better than the baseline method when considering the mean and median of the distributions. We also find that for those categories, our method produces more experiments with 0 errors.

One advantage of our method is that it generates less plans with missing actions compared with the baseline. As XDL generation in SynthReader is implemented by rule-based pattern-matching techniques, any actions that do not match those templates would not appear in the final XDL. For example, for the protocol:

```
To a solution of m-CPBA (200 mg, 0.8 mmol)
in dichloromethane (10 mL), cooled to 0 °C,
was added dropwise a solution of 5-chloro
-10-oxa-3-thia-tricyclo [5.2.1.01, 5] dec-8-
ene (150 mg, 0.8 mmol) in dichloromethane
(10 mL).
```

the plan generated by CLAIRIFY was

```
<Add vessel="V1" reagent="m-CPBA" amount="
200 mg"/>
<Add vessel="V1" reagent="dichloromethane"
volume="10 mL"/>
<HeatChill vessel="V1" temp="0 °C" time="3
min"/>
<Add vessel="V2" reagent="5-chloro-10-oxa-3-
thia-tricyclo[5,2,1,0*1,5*]dec-8-ene" amount
="150 mg"/>
<Add vessel="V2" reagent="dichloromethane"
volume="10 mL"/>
<Transfer from_vessel="V2" to_vessel="V1"/>
```

while the plan generated from SynthReader was

```
<Add vessel="reactor" reagent="5-chloro-10-
oxa-3-thia-tricyclo" volume="0" speed="40.0"
/>
```

Our model is able to decompose a complicated procedure into simpler actions by making two solutions in separate beakers and combining them with a Transfer procedure. It also assumes that the solutions don't already exist as mixtures and creates them from scratch. This is another benefit of our model, as it is able to understand implicit actions. For example, given the prompt

```
L-Ornithine (31.92 g, 120 mmol) was added to
 a mixture of KOH (6.72 g, 120 mmol), water
(200 ml) and THF (100 ml)
```

SynthReader considers a mixture of three chemicals as a single solution and creates the action:

```
<Add vessel="reactor" reagent="a mixture of
KOH (6.72 g, 120 mmol), water (200 ml) and
THF (100 ml)" volume="0" speed="40.0"/>
<AddSolid vessel="reactor" reagent="L-
Ornithine" mass="31.92 g"/>
```

On the other hand, CLAIRIFY correctly understand the implicit action to mix them beforehand and generates an appropriate XDL:

```
<Add vessel="V1" reagent="L-Ornithine"
amount="31.92 g"/>
<Add vessel="V1" reagent="KOH" amount="6.72
g"/>
<Add vessel="V1" reagent="Water" amount="200
 ml"/>
<Add vessel="V1" reagent="THF" amount="100
ml"/>
```

However, our model produced plans with a greater number of wrong actions than SynthReader. This is likely because our model is missing domain knowledge on certain actions that would need to be included in the prompt or verifier. For example, given the instruction *"Dry solution over magnesium sulfate"*, our model inserts a `<Dry.../>` into the XDL plan, dbut the instruction is actually referring to a procedure where one passes the solution through a short cartridge containing magnesium sulphate, a procedure which seems to be encoded in SynthReader. Another wrong action our model performs is reusing vessels. In chemistry, one needs to ensure a vessel is uncontaminated before using it. However, our model generates plans that can use the same vessel in two different steps without washing it in between. Our model also sometimes generates plans with ambiguous values. For example, many experiment descriptions include conditional statements such as *"Heat the solution at the boiling point until it becomes white"*. Conditions in XDL need a numerical condition as a parameter. Our model tries to incorporate them by including actions such as `<HeatChill temp="boiling point" time = "until it becomes white"/>`, but they are ambiguous. We can make our model better in the future by incorporating more domain knowledge into our structured language description and improving our verifier with real-world constraints. For example, we can incorporate visual feedback from the environment, include look-up tables for common boiling points, and ensure vessels are not reused before cleaning. Other errors include misunderstanding notations commonly used in chemistry experiments. For instance, instructions such as *"Wash with EtOAc (2 × 10 mL)"* indicate the need for two separate `WashSolid()` actions in XDL. However, the large language model often struggles to parse the *(2 × 10 mL)* notation correctly, resulting in either performing a single `WashSolid()` action with 10 mL (or occasionally 20 mL if a multiplication action is inferred) or (more rarely) omitting the action entirely.

Another cause of errors occurs when a specific value is not provided in the instruction. For example, if an instruction states *"wash product in EtOAc"* without specifying a volume, the model is unable to generate a reasonable value and defaults to writing 0 mL in the generated plan.

Despite the XDL plans generated by our method containing errors, we found that the experts placed greater emphasis on missing actions than ambiguous or wrong actions when picking the preferred output, indicating larger severity of this class of error for the tasks and outputs investigated here.

Chem-EDU dataset We annotated the errors in the Chem-EDU datasets using the same annotation labels as for the Chem-RnD dataset. The breakdown of the errors is in the right plot of Fig. 8. Note that we did not perform a comparison with SynthReader as no plans were generated from it. We find that the error breakdown is similar to that from Chem-RnD, where we see amibiguous values in experiments that have conditionals instead of precise values. We also encounter a few wrong parameter errors, where the model does not

Table 2 Verifier analysis

Dataset	Average num. verifier calls	Max/min verifier calls	Error type caught by verifier [count]
Chem-RnD	2.58 ± 2.00	10/1	Missing property in action [306]
			Property not allowed [174]
			Wrong tag [120]
			Action does not exist [21]
			Item not defined in hardware or reagents list [15]
			Plan cannot be parsed as XML [6]
Chem-EDU	1.14 ± 0.47	3/1	Item not defined in hardware or reagents list [47]
			Property not allowed [26]
			Wrong tag [40]
			Missing property in action [3]

We report the average number of times CLAIRIFY calls the verifier for the experiments in a given dataset, as well as the minimum and maximum number of times. We also report the type of error encountered by the verifier and the number of times it caught that type

include units for measurements. This can be fixed in future work by improving the verifier to check for these constraints.

3. Number of interventions required by the verifier. To better understand the interactions between the generator and verifier in CLAIRIFY, we analyzed the number of interactions that occur between the verifier and generator for each dataset to understand the usefulness of the verifier. In Table 2, we show that each experiment in the Chem-RnD dataset runs through the verifier on average 2.6 times, while the Chem-EDU dataset experiments runs through it 1.15 times on average. The difference between the two datasets likely exists because the Chem-EDU experiments are shorter and less complicated. The top Chem-EDU error encountered by the verifier was that an item in the plan was not defined in the Hardware or Reagents list, mainly because we included hardware constraints for this dataset that we needed to match in our plan. In Fig. 9, we show a sample loop series between the generator and verifier.

4.2 Robot execution

To analyze how well our system performs in the real world, we execute a few experiments from the Chem-EDU dataset on our robot. Three experiments from the Chem-EDU dataset were selected to be executed.

4.2.1 Experiment setup

Hardware The proposed lab automation framework has been evaluated using the Franka Emika Panda arm robot, equipped with a Robotiq 2F-85 gripper and an Intel RealSense D435i stereo camera mounted on the gripper to allow for active vision. The robot's DoF has been extended by one degree (in total 8-DoF) at its end-effector using a Dynamixel XM540-W150 servo motor. Figure 2 shows the hardware setup.

Lab tools integration The robot framework is expanded by incorporating lab tools. We used an IKA RET control-visc device, which works as a scale, hotplate, and stir plate, and a Sartorius BCA2202-1 S Entris, which works as a high-precision weighing scale. The devices communicate with the TAMP solver to execute chemistry specific skills.

Software The robot is controlled using FrankaPy (Zhang et al., 2020). We implemented a ROS wrapper for the servo motor (8th DoF). To detect fiducial markers, we use the AprilTag library (Olson, 2011). We use the MoveIt motion planning framework (Coleman et al., 2014) for our TAMP solver and its streams. The constrained planning function (Kingston et al., 2019) is an extension of elion.[3]

4.2.2 Solution color change based on pH

As a basic chemistry experiment, we demonstrated the color change of a solution containing red cabbage juice. This is a popular introductory demonstration in chemistry education, as the anthocyanin pigment in red cabbage can be used as a pH indicator (Fortman & Stubbs, 1992). We prepared red cabbage solution by boiling red cabbage leaves in hot water. The colour of the solution is dark purple/red. Red cabbage juice changes its color to bright pink if we add an acid and to blue if we add a base, and so we acquired commercially-available vinegar (acetic acid, an acid) and baking soda (sodium bicarbonate, a base).

[3] https://github.com/JeroenDM/elion.

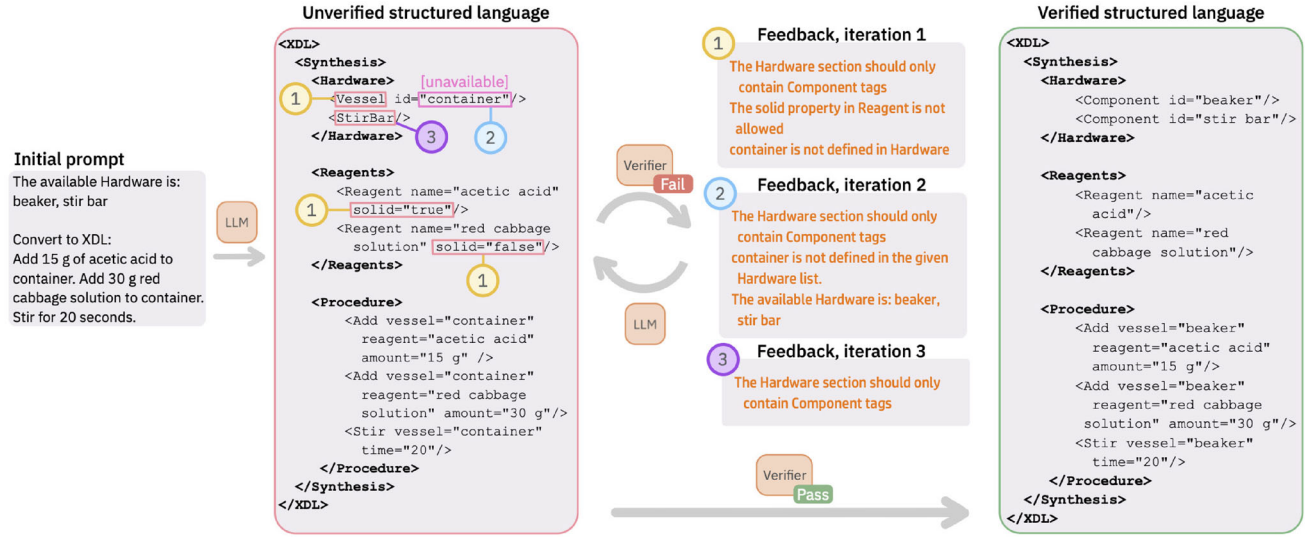


Fig. 9 Feedback loop between the Generator and Verifier. The input text is converted to structured-like language via the generator and is then passed through the verifier. The verifier returns a list of errors (marked with a yellow 1). The feedback is passed back to the generator along with the erroneous task plan, generating a new task plan. Now that previous errors were fixed and the tags could be processed, new errors were found (including a constraint error that the plan uses a vessel not in the environment). These errors are denoted with a blue 2. This feedback loop is repeated until no more errors are caught, which in this case required 3 iterations (Color figure online)

In this experiment, we generated XDL plans using CLAIRIFY from two language inputs:

```
[1] Add 40 g of red cabbage solution into a
beaker. Add 10 g of acetic acid into the
beaker, then stir the solution for 10
seconds.
```

```
[2] Add 40 g of red cabbage solution into a
beaker. Add 10 g of baking soda into the
beaker, then stir the solution for 10
seconds.
```

Figure 10 shows the flow of the experiment. Our system generated a XDL plan that correctly captured the experiment; the plan was then passed through TAMP to generate a low-level action plan and was then executed by the robot.

4.2.3 Kitchen chemistry

We then tested whether our robot could execute a plan generated by our model for a different application of household chemistry: food preparation. We generated a plan using CLAIRIFY for the following lemonade beverage, which can be viewed on our website:

```
Add 15 g of lemon juice and sugar mixture to
 a cup containing 30 g of sparkling water.
Stir vigorously for 20 sec.
```

Figure 11 shows the flow of an experiment. Since we deal with edible material, we implemented a different stirring motion that does not touch the content of the container.

4.2.4 Solubility measurement

We finally measured the solubility of household solutes as an example of basic educational chemistry experiments for students (Wolthuis et al., 1960). Measuring solubility has desirable characteristics as a benchmark for automated chemistry experiments: (i) it requires basic chemistry operations, such as pouring, solid dispensing, and observation of the solution status, (ii) solubility can be measured using ubiquitous food-safe materials, such as water, salt, sugar, and (iii) the accuracy of the measurement can be evaluated quantitatively by comparing with literature values. We measured the solubility of three solutes: table salt (sodium chloride), sugar (sucrose), and alum (aluminum potassium sulfate).

The robot estimates the amount of water to make a saturated solution by repeatedly pouring a small amount of water. After pouring, the solution is stirred and the turbidity before and after stirring was compared. The turbidity decreases by stirring if the remaining solutes dissolved into water, whereas it stays at a constant value if there are no residues. If the turbidity decrease after the N-th pouring is smaller than 5%, we assume there were no residues at the beginning of N-th pouring and that the amount of water required to dissolve all solutes is between the volume of water added at the $(N-2)$-th and $(N-1)$-th pouring. We use the average of the two for simplicity of presentation. Figure 13 shows an example of turbidity change during the experiment.

A natural language explanation for the above solubility measurement protocol is as follows:

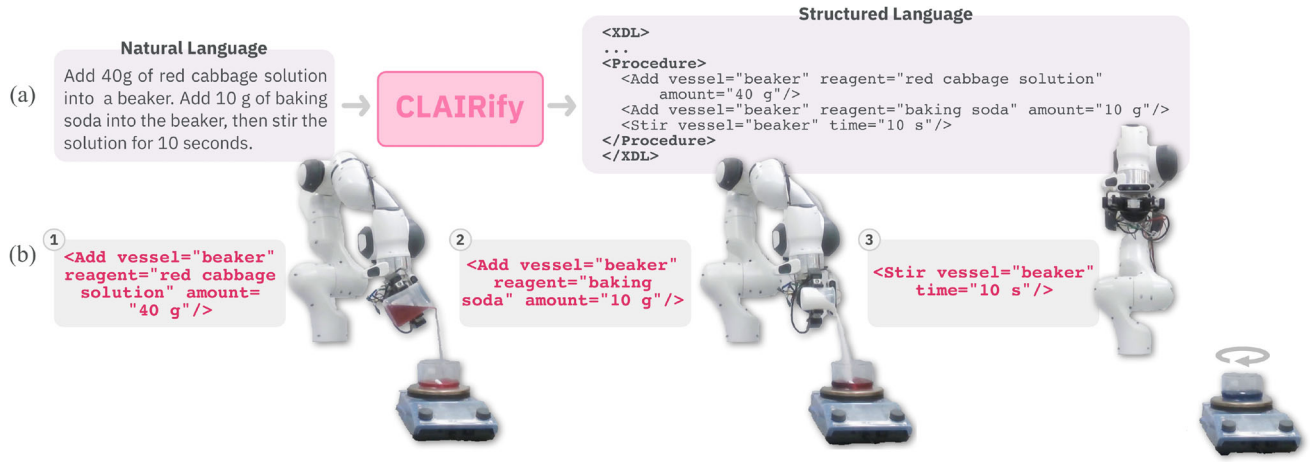


Fig. 10 Robot execution: The robot executes the motion plan generated from the XDL for given natural language input. **a** CLAIRIFY converts the natural language input from the user into XDL. **b** The robot interprets XDL and performs the experiment. Stirring is done by a rotating stir bar inside the beaker

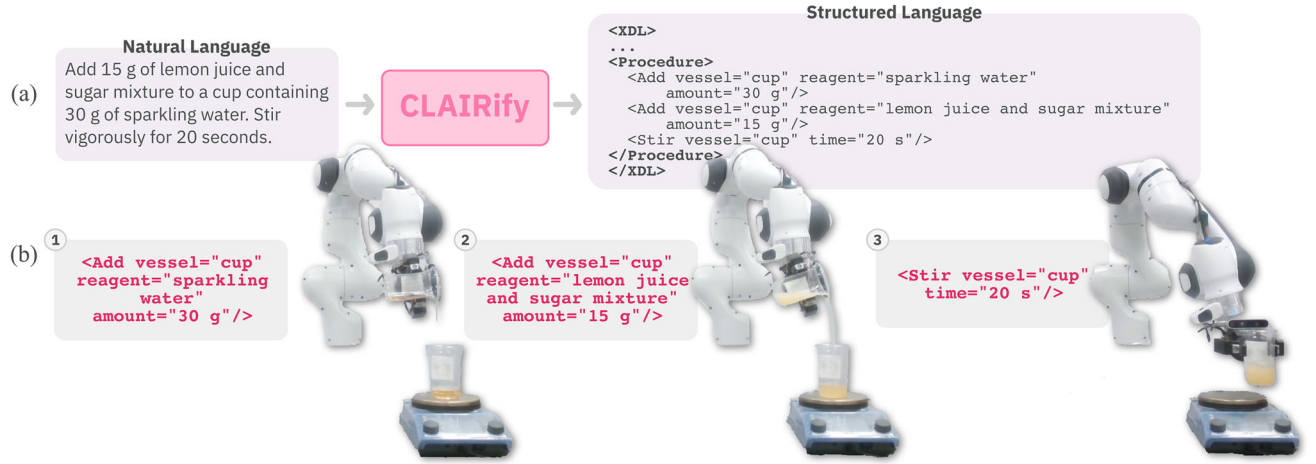


Fig. 11 Kitchen chemistry: The robot executes the motion plan generated from the XDL for given natural language input. **a** CLAIRIFY converts the natural language input from the user into XDL. **b** The robot interprets XDL and performs the experiment. Stirring is done by shaking a cup with the robot arm

```
Add 10 g of salt to the beaker.
Repeat the following steps for five times.
Add 10 g of water into the beaker, and
measure the turbidity.
After stirring for 90 seconds, measure the
turbidity again.
```

Note that we extended the XDL to allow turbidity as a measurable quantity of `<Monitor>` since the XDL standard at the time of writing (XDL 2.0.1) only supports temperature and pH. We added this skill to our defintion of XDL that we input to the LLM in CLAIRIFY. The amount of solute and stirring time were changed for different solutes. The workflow of the solubility experiments is shown in Fig. 12.

The measured solubility for three solutes is shown in Table 3. The robot framework managed to measure the solubility with sufficient accuracy that they are comparable to solubility values found in the literature (NAOJ, 2022).

The primary reason for the difference from the literature value is the range of minimum amount of water required for dissolving. In an example of turbidity change shown in Fig. 13, the robot can only tell the second pouring is insufficient and the third pouring is sufficient to dissolve all solutes, but it cannot tell the exact required amount. As a result, the solubility measurement inherently includes error caused by the resolution of pouring. We can reduce the error by pouring a smaller amount of water at once, but pouring less than 10 g is difficult because of the delayed feedback of the scale and the scale minimum resolution. We can improve the accuracy of solubility measurements by developing a pipette designed for a robot.

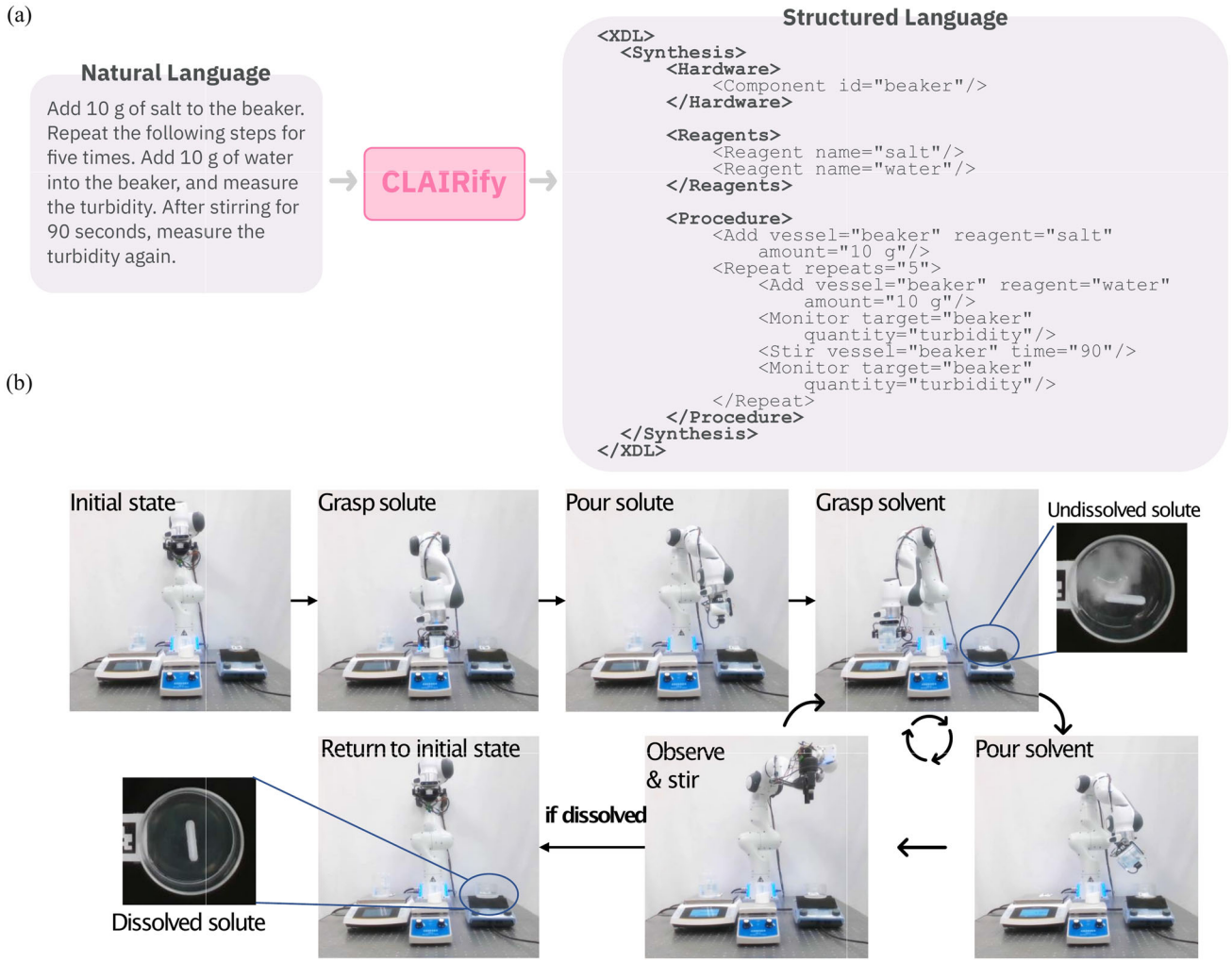


Fig. 12 Workflow of solubility experiment. **a** We translate a natural language input to XDL using CLAIRify. **b** We then execute the plan on a robot using TAMP. First, a fixed amount of the solute is added to the dish on the weighing scale and stirrer. The robot pours 10 g of water into the dish. The solution is mixed with the magnetic stirrer. After stirring, the turbidity of the solution is measured to check dissolvement. If undissolved, another 10 g of water is added until no solutes remain. The experiment was conducted at room temperature (25°C)

Table 3 Results of the solubility experiments

Solute	Solute (g)	Water (g)	Solubility	Lit.	% error
Salt	13.9	41.8	33.2	35.8	7.2
Sugar	60.00	26.46	226.8	203.9	11.2
Alum	3.00	29.87	10.0	11.4	12.3

Amount of solute in the beaker, amount of water to dissolve all solute, calculated solubility (the amount of solute dissolved per 100 g of water), and literature data (lit.) for solubility at 20°C is shown. Literature data are taken or calculated from (NAOJ, 2022)

4.2.5 Recrystallization experiment

Recrystallization is a purifying technique to obtain crystals of a solute by using the difference in solubility at different temperatures. Typically, solutes have higher solubility at high temperatures, meaning hot solvents will dissolve more solute than cool solvents. The excess amount of solute that cannot be dissolved anymore while cooling the solvent precipitates and forms crystals. We tested the recrystallization of alum by changing the temperature of the water. Alum was chosen as the target solute since its solubility greatly changes according to water temperature. The recrystallization experiment setup extends the solubility test by pre-heating the solvent. A natural language explanation for the above recrystallization experiment protocol is as follows:

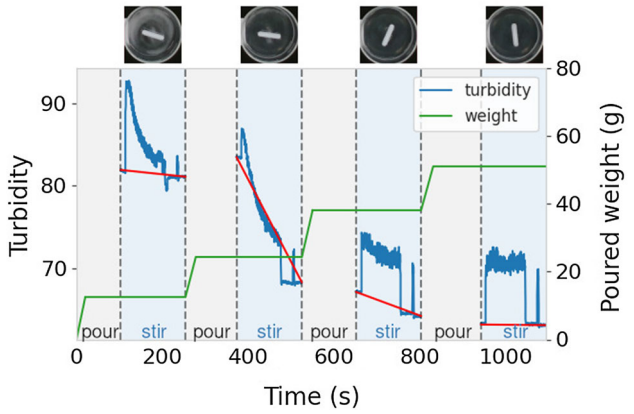


Fig. 13 Turbidity change during experiment. Water is poured into the dish during pouring (grey) and turbidity is measured during observation (blue). The end of the experiment is determined by turbidity comparison. In this example, all solutes are dissolved at the third pouring because the turbidity change after the fourth pouring is below the threshold. The average weight of the second and third pouring is used to calculate the solubility (Color figure online)

Fig. 14 Recrystallization of alum inside the water. After heating water by putting a beaker with water on a hotplate, the robot poured alum into a dish. The robot then poured hot water, and the solution was heated and stirred. The formation of a precipitate is observed after the dish is cooled down. The dried crystals in a vial are shown

```
Add 50g of water to beaker.
Heat the beaker filled with water for 1 min
to 60 C.
Add 20 g of alum into an empty beaker, and
add 50 g of the heated water into the beaker.
Cool the beaker for 30 min to 20 C.
```

Figure 14 shows the result of the experiment.

4.3 Ablation studies

We assess the impact of various components in our prompt designs and feedback messaging from the verifier. We performed these tests on a small validation set of 10 chemistry experiments from Chem-RnD (not used in the test set) and report the number of XDL plans successfully generated (i.e., was not in the iteration loop for $x = 10$ steps).

4.3.1 Prompt design

To evaluate the prior knowledge of the GPT-3 on XDL, we first tried prompting the generator without a XDL description, i.e., with the input:

```
initial_prompt = """
Convert to XDL:
# <Natural language instruction>"""
```

The LLM was unable to generate XDL for any of the inputs from the small validation set that contains 10 chemistry experiments. For most experiments, when asked to generated XDL, the model output a rephrased version of the natural language input. In the best case, it output some notion of structure in the form of S-expressions or XML tags, but the outputs were very far away from correct XDL and were not related to chemistry. We tried the same experiment with `code-davinci-002`; the outputs generally had more structure but were still nonsensical. This result suggests the LLM does not have the knowledge of the target language and including the language description in the prompt is essential to generate an unfamiliar language.

4.3.2 Feedback design

We experimented with prompts in our iterative prompting scheme containing various levels of detail about the errors. The baseline prompt contains a description as well as the natural language instruction. We wanted to investigate how much detail is needed in the error message for the generator to be able to fix the errors in the next iteration. For example, is it sufficient to write “There was an error in the generated XDL", and do we need to include a list of errors from the verifier (such as “Quantity is not a permissible attribute for the Add tag"), or do we also need to include the erroneous XDL from the previous iteration? We also wanted to keep any feedback messages as general as possible to reduce prompt lengths, considering there is a limit for how many tokens we can query OpenAI models with.

The XDL generation success rate for different error message designs is shown in Table 4. We find that including the erroneous XDL from the previous iteration and specifying why it was wrong resulted in the highest number of successfully generated XDL plans. Including a list of errors was better than only writing “This XDL was not correct. Please fix the errors”, which was not informative enough to fix any errors. Including the erroneous XDL from the previous iteration is also important; we found that including only a list of the errors without the context of the XDL plan resulted in low success rates.

4.4 Component analysis for robot execution

4.4.1 Pouring skill evaluation

We evaluated the accuracy and efficiency of the pouring skill for liquid and powder. To evaluate the effect of our proposed pouring method, we implemented a PD control pouring method where end-effector angular velocity is proportional to

Table 4 Number of XDL plans successfully generated for different error message designs in the iterative prompting scheme on a validation set from Chem-RnD consisting of 10 experiments

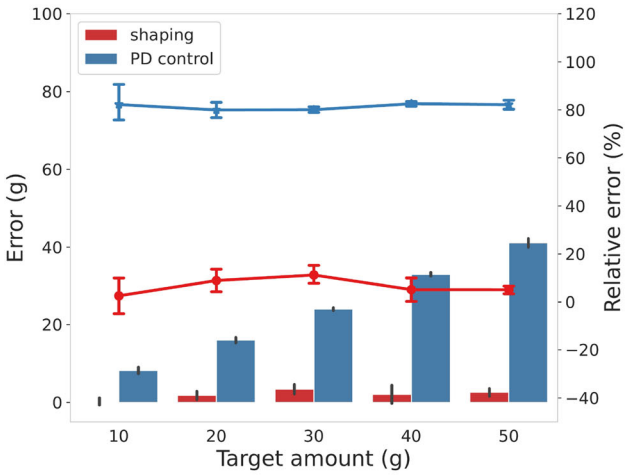
Variations of iterative prompt design using verifier error messages	Plan's generated success rate (%)↑
Naive: XDL from previous iteration and string "This XDL was not correct. Please fix the errors."	0
Last error: Error list from verifier from previous iteration	30
All errors cumulative: Accumulated error list from all previous iterations	50
XDL + Last error: XDL and error list from verifier from previous iteration	100

the difference between target and feedback weight as a baseline. Figure 15 shows the pouring experiment results. The results show that the shaping function contributed to reducing the overshooting compared to PD control pouring. The overshoot of the PD control pouring is mainly because of the scale's delayed feedback (∼3 s). The intermittent pouring caused by the shaping function compensated for the delay and improved the overall pouring accuracy. On average, the pouring errors using the shaping approach for water and salt were 2.2 ± 1.5 g (8.1 ± 4.8 %) and 2.1 ± 1.4 g (8.8 ± 6.2 %). The errors for PD control were 24.5 ± 12.0 g (81.4 ± 4.5 %) and 7.7 ± 4.3 g (24.1 ± 5.2 %). Moreover, as we can see both the error and relative error stay approximately constant with respect to the target amount when using the shaping method, in contrast to the PD controller. The average pouring times with the shaping function for 50 g water and salt were 25.1 s and 36.8 s, respectively. Our results are comparable with previous work (Kennedy et al., 2019; Huang et al., 2021) in terms of pouring error and time, without using a learned, vision-based policy, or expensive equipment setup.

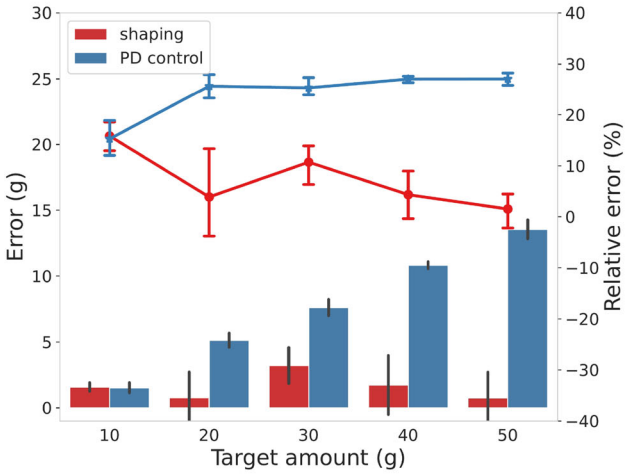
In order to assess the pouring skill more effectively, a series of experiments were carried out involving three human subjects. The experiments aimed to measure both the accuracy and speed of pouring, with each experiment being repeated five times. The average errors for 50 g of water and salt were 0.8 g and 1.0 g, and the average pouring times were 16.4 s and 18.6 s respectively. The main factor causing the disparity in pouring time between human subjects and the robot stems from their distinct behaviors during the pre-pouring phase. Humans have the ability to quickly rotate a beaker during the pre-pouring stage by relying on visual cues to determine when the pouring begins. Conversely, the robot's movements during the pre-pouring stage were deliberately slow to prevent overshooting. On average, the pre-pouring stage of the robot took approximately 6 s, whereas it was negligible for humans.

4.4.2 Constrained motion planning in 7/8-DoF robot

The constrained motion planning performance of 7-DoF and 8-DoF robots is evaluated in two scenarios: (1) single step, (2) two steps. In scenario (1), robots find a constrained path with a fixed orientation from initial to final positions that are randomly sampled. Scenario (2) extends the first with an additional intermediate sampled waypoint. For each scenario, we run 50 trials in Algorithm 3 with random seeding of the IK solver.



(a) Pouring error in water.



(b) Pouring error in salt.

Fig. 15 Evaluation of pouring error. The pouring errors of our shaping pouring and PD control baseline pouring are compared using **a** water and **b** salt. The bar plot shows the error (poured amount—target amount) and the line plot shows the relative error. The error bars show the standard deviation (Color figure online)

Table 5 Success rate comparison of 7 and 8-DoF robot

	Scenario 1 (%)		Scenario 2 (%)	
	IK	Plan	IK	Plan
7-DoF	99	84	99	70
8-DoF	100	97	100	84

In scenario (2), we restart the sequence planning from the first step if a step fails. Constraints are set to the robot end-effector pitch and roll ($\|\theta, \phi\| \leq 0.1$ rad).

The performance of the 7-DoF and 8-DoF robot arms for the two scenarios are shown in Table 5. The results show that the IK and constrained motion planning have higher success rates in 8-DoF compared with the 7-DoF robot.

5 Discussion

In this paper, we demonstrate how LLMs are effective tools for translating natural language inputs into domain-specific target languages without any fine-tuning. We find that by prompting an LLM with errors that it makes, it is able to correct its own output and generate syntactically valid plans. We also tested the ability of CLAIRIFY to generate plans for the same experiment written in different ways (see "Appendix D"), which is important for generalizability since humans have different writing styles. We find that LLMs are robust to variations in natural language, which is important for lowering the barrier to successful user interaction. The XDL plans generated by CLAIRIFY can then be combined with our TAMP pipeline to effectively perform multistep chemistry experiments in the real world. However, the current study is limited to a few types of chemistry experiments because the number of skills incorporated in the framework is limited. Increasing the repertoire of skills, such as glassware perception in 3D and clutter, without fiducial markers (Eppel et al., 2020), can improve the framework scalability. As we develop more and more skills, we can append their descriptions to the language model input. We demonstrated this by appending a new skill, `<Monitor>`, to the XDL description, and the LLM was able to accurately incorporate it into the plan. Another issue is that the inefficiency of PDDLStream inhibits the framework from being reactive in a dynamic environment. Incorporating the learning-based search heuristics for PDDLStream (Khodeir et al., 2023, 2022) may overcome this limitation. Constrained motion planning was shown to effectively avoid spillage of the beaker contents during transfer in our experiments. We have also shown that adding an extra 8th DoF to the robot enabled more flexibility and a higher success rate for constrained motion planning. However, the proposed constrained motion planning embedded in TAMP cannot run in real-time. Considering the dynamics of the beaker content may help to have higher flexibility in robot manipulation (Muchacho et al., 2022). Although our skill has currently attained 8% error for liquid and powder pouring, higher accuracy is desirable for precise experiments in a chemistry lab. We used a scale with integrated functionality for stirring and heating, but its measurement is delayed for 3 s. Higher precision pouring can be attained using a scale with a shorter response time; also, it can be achieved by specialized tools, such as a pipette (Yoshikawa et al., 2023). In addition, visual feedback during pouring may lead to faster and more accurate pouring and be helpful in avoiding spillage.

Moreover, CLAIRIFY was successful in generating plans beyond the state-of-the-art method for the chemistry domain-specific structured language XDL. Although the generated plans were syntactically correct and satisfied the constraints, they contained errors. However, experts placed greater emphasis on missing actions than on ambiguous or incorrect actions when selecting the preferred output, indicating that this class of error is more severe for the tasks and outputs investigated here. These results demonstrate the generalizability of our method, which uses zero-shot iterative prompting verification. We can apply it to different language styles and domains and still obtain coherent plans. While our approach, which combines LLMs and TAMP, showed promising results in generating feasible and executable plans, as evidenced by our evaluation, the capabilities of pure LLMs in generating semantically correct plans remain limited. The limitation in task planning abilities has been highlighted in a recent study (Bubeck et al., 2023) as well. To address this shortcoming, an alternative approach could be to incorporate human-in-the-loop planning or to utilize multi-modal foundation model that consider the surrounding scene of the robot.

Another important consideration to address is the tradeoff between the human interpretability and expressive power of the target structured language. Our approach to using intermediate language enables users to ensure the LLM's natural language interpretation is reasonable; however, the expressive power of XDL imposed limitations on the framework's abilities. The robot framework can conduct more diverse actions than XDL can express, but the use of XDL limits the available actions. This problem may be alleviated by generating the robot program directly, but human interpretability may be decreased as a result.

6 Conclusions

In this paper, we presented a framework for automating chemistry lab experiments using general-purpose robot manipulators and natural language commands. In order to facilitate the closed-loop execution of long-horizon chemistry experiments, CLAIRIFY maps natural language com-

mands to XDL, a human-interpretable intermediate language that standardizes chemistry experiment descriptions. Subsequently, XDL instructions are converted into a sequence of subgoals for a constrained task and motion plan solver, and the robot executes those plans using its diverse set of skills. Finally, the robot visually monitors the progress of the tasks. We demonstrated that our approach lowers the barriers to instructing robots by non-experts to execute robot task plans. The robot handles solubility and recrystallization experiments autonomously when provided with natural language inputs.

Acknowledgements We thank members of the Matter Lab for annotating task plans. We would also like to thank the Acceleration Consortium and Google Inc. for their generous support (NSERC-Google Industrial Research Chair Award). L.B.K. acknowledges generous support from the Carlsberg Foundation.

Author Contributions NY, MS, KD, SA-R, ZJ, LBK worked on CLAIRify; NY, KD, AZL, YZ, HX, AK worked on robotic experiments. NY, MS, KD wrote the initial draft. LBK, AA-G, FS, AG proof-read the manuscript, and AA-G, FS, AG supervised the project.

Funding L.B.K. has received funding from the Carlsberg Foundation under grant ID CF21-0669.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interest to disclose.

Appendix A: Pouring policy

The pouring policy is the blending of a shaping function $s(t)$ and a model-free PD controller $\boldsymbol{v}_{\mathrm{PD}}(t)$, expressed as $\boldsymbol{v}_{\mathrm{PD}}(t) \times s(t)$. The PD controller is defined as:

$$\boldsymbol{v}_{\mathrm{PD}}(t) = \boldsymbol{k}_p e + \boldsymbol{k}_d \dot{e}, \tag{1}$$

where $e(t) = x_{ref} - x_{fb}$. The shaping function is implemented via the summation of several unit functions $u(t)$ as follows:

$$\begin{aligned} s(t) = {} & u(t) - 2\,u(t - t_0) \\ & + \textstyle\sum_{k=1}^{N} \{u(t - (t_0 + k\,T_{deactive})) \\ & + u(t - (t_0 + k\,(T_{deactive} + T_{idle}))) \\ & - 2\,u(t - (t_0 + k\,(T_{deactive} + T_{idle} + T_{activate})))\} \end{aligned} \tag{2}$$

where t_0 is the moment in which $e(t)$ starts to change, meaning that the material is getting added to the target dish. $T_{deactive}$, T_{idle}, and T_{active} are the parameters set by the user to describe the periodic motion of the robot end-effector. This motion continues till the material transferred to the target dish reaches the desired amount.

Appendix B: Full prompt

The full XDL description and an experiment description from the Chem-EDU dataset are shown as an example of a full prompt to the LLM.

```
initial_prompt = """
XDL files will follow XML syntax and consist
 of three mandatory sections: Hardware,
where virtual vessels that the reaction
mixture can reside in are declared. Reagents
, where all reagents that are used in the
procedure are declared, and Procedure, where
 the synthetic actions involved in the
procedure are linearly declared.

XDL File Stub:
<XDL>
  <Synthesis>
      <Hardware>
          <!-- ... -->
      </Hardware>

      <Reagents>
          <!-- ... -->
      </Reagents>

      <Procedure>
          <!-- ... -->
      </Procedure>
  </Synthesis>
</XDL>

Hardware:
Each individual reagent, unless otherwise
stated should be contained within their own
vessels.

(format is(Property, Type, Description))

id, str, Name of hardware

Reagents:
The Reagents section contains Reagent
elements with the props below.
Any reagents which were combined before the
experiment should be combined as one reagent
 before the procedure. (i.e. 'lime juice
mixed with sugar' = <Reagent name='lime
juice mixed with sugar')

Reagent:
Reagent used by procedure.

(format is(Property, Type, Description))

name, str, Name of reagent
```

```
Procedure:
All steps included in the Full Steps
Specification may be given within the
Procedure block of a XDL file. Additionally,
 the Procedure block may be, but does not
have to be, divided up into Prep, Reaction,
Workup and Purification blocks, each of
which can contain any of the steps in the
specification.

Here is a list of tags that can be used in
this language:
Liquid Handling: Add, Separate, Transfer,
Stirring: StartStir, Stir, StopStir,
Temperature Control: HeatChill,
HeatChillToTemp, StartHeatChill,
StopHeatChill
Inert Gas: EvacuateAndRefill, Purge,
StartPurge, StopPurge
Filtration: Filter, FilterThrough, WashSolid
Special: Wait, Repeat,
Other: CleanVessel, Crystallize, Dissolve,
Dry, Evaporate, Irradiate, Precipitate,
ResetHandling, RunColumn

Steps:
Liquid Handling:
Add liquid or solid reagent. Reagent
identity (ie liquid or solid) is determined
by the solid property of a reagent in the
Reagent section.

The quantity of the reagent can be specified
 using either volume (liquid units) or
amount (all accepted units e.g. 'g', 'mL', '
eq', 'mmol').

format(Property Type    Description)
vessel  vessel  Vessel to add reagent to.
reagent reagent Reagent to add.

Separate:
Perform separation.
format(Property Type Description)
Property    Type    Description
purpose str 'wash' or 'extract'. 'wash'
means that product phase will not be the
added solvent phase, 'extract' means product
 phase will be the added solvent phase. If
no solvent is added just use 'extract'.
product_phase   str 'top' or 'bottom'. Phase
 that product will be in.
from_vessel vessel  Contents of from_vessel
are transferred to separation_vessel and
separation is performed.
separation_vessel   vessel  Vessel in which
separation of phases will be carried out.
to_vessel   vessel  Vessel to send product
phase to.

Transfer:
Transfer liquid from one vessel to another.

The quantity to transfer can be specified
using either volume (liquid units) or amount
 (all accepted units e.g. 'g', 'mL', 'eq', '
mmol').

format(Property Type    Description)
from_vessel vessel  Vessel to transfer
liquid from.
to_vessel   vessel  Vessel to transfer
liquid to.

Stirring:
StartStir:
Start stirring vessel.
format(Property Type    Description)
vessel  vessel  Vessel to start stirring.

Stir:
Stir vessel for given time.
format(Property Type    Description)
vessel  vessel  Vessel to stir.
time    float   Time to stir vessel for.

StopStir:
Stop stirring given vessel.
format(Property Type    Description)
vessel  vessel  Vessel to stop stirring.

Temperature Control:
HeatChill:
Heat or chill vessel to given temp for given
 time.
format(Property Type    Description)
vessel  vessel  Vessel to heat or chill.
temp    float   Temperature to heat or chill
 vessel to.
time    float   Time to heat or chill vessel
 for.

Convert to XDL:
Add 30 g of red cabbage soup into a beaker.
Add 10 g of acetic acid into the beaker,
then stir the solution for 10 seconds.
"""
```

Appendix C: Chem-EDU dataset

We show some examples of experiments in the Chem-EDU dataset.

```
Add 30 g of red cabbage soup into a beaker.
Add 10 g of acetic acid into the beaker,
then stir the solution for 10 seconds.
```

Experiment 0

```
Acacia Honey Syrup
Add 200g of hot water to a beaker. Add 430g
acacia honey to the beaker. Stir for 10
minutes while heating.
```

Experiment 13

```
Add 250g water to beaker. Heat until it
reaches 100 degrees C. Add 40 g of pasta.
Heat for 6 minutes at 100 degrees C.
```

Experiment 30

Appendix D: Variations in natural language

Since humans have different writing styles, we wanted to determine if our model is able to generate consistent XDL plans from natural language descriptions written in different ways. We took two experiments from Chem-EDU and prompted ChatGPT to rewrite the experiment. We ran all experiments through CLAIRIFY and examined the outputs.[4] We find that CLAIRIFY can generate the same XDL plans with different natural language descriptions.

[4] https://beta.openai.com.

```
Add 30 g of red cabbage soup into a beaker. Add 10 g of acetic acid into the beaker, then stir the
solution for 10 seconds.
```

(a) Experiment 0 - natural language input in writing style 1

```
<XDL>
  <Synthesis>
      <Hardware>
          <Component id="beaker" />
      </Hardware>

      <Reagents>
          <Reagent name="red cabbage soup" />
          <Reagent name="acetic acid" />
      </Reagents>

      <Procedure>
          <Add vessel="beaker" reagent="red cabbage soup" amount="30 g" />
          <Add vessel="beaker" reagent="acetic acid" amount="10 g" />
          <Stir vessel="beaker" time="10 s" />
      </Procedure>
  </Synthesis>
</XDL>
```

(b) Experiment 0 - XDL plan generated by CLAIRify for writing style 1

```
Pour 30 grams of red cabbage soup into a beaker. Introduce 10 grams of acetic acid into the same
beaker and proceed to stir the solution for a duration of 10 seconds.
```

(c) Experiment 0 - natural language input in writing style 2 (generated using ChatGPT)

```
<XDL>
  <Synthesis>
      <Hardware>
          <Component id="beaker" />
      </Hardware>

      <Reagents>
          <Reagent name="red cabbage soup" />
          <Reagent name="acetic acid" />
      </Reagents>

      <Procedure>
          <Add vessel="beaker" reagent="red cabbage soup" amount="30 g" />
          <Add vessel="beaker" reagent="acetic acid" amount="10 g" />
          <StartStir vessel="beaker" />
          <Stir vessel="beaker" time="10 s" />
          <StopStir vessel="beaker" />
      </Procedure>
  </Synthesis>
</XDL>
```

(d) Experiment 0 - XDL plan generated by CLAIRify for writing style 2

Fig. 16 XDL generated for different writing styles for Experiment 0 from Chem-EDU. The first writing style (writing style 1) is taken directly from Chem-EDU. For the second writing style (writing style 2), we asked ChatGPT to rewrite it. For both prompts, we generate a XDL plan. The output is semantically the same for both (note that the extra `<StartSir>` and `<StopStir>` actions in (d) are redundant, as the presence of the `<Stir>` action automatically does a `<StartSir>` and `<StopStir>` action)

```
Add 250g water to beaker. Heat until it reaches 100 degrees C. Add 40 g of pasta. Heat for 6 minutes
at 100 degrees C.
```

(a) Experiment 30 - natural language input in writing style 1

```
<XDL>
  <Synthesis>
      <Hardware>
          <Component id="beaker" />
      </Hardware>

      <Reagents>
          <Reagent name="water" />
          <Reagent name="pasta" />
      </Reagents>

      <Procedure>
          <Add vessel="beaker" reagent="water" amount="250g" />
          <HeatChill vessel="beaker" temp="100" time="0" />
          <Add vessel="beaker" reagent="pasta" amount="40g" />
          <HeatChill vessel="beaker" temp="100" time="6" />
      </Procedure>
  </Synthesis>
</XDL>
```

(b) Experiment 30 - XDL plan generated by CLAIRify for writing style 1

```
In a beaker, pour 250 grams of water. Apply heat until the water reaches a temperature of 100 degrees
Celsius. Then, carefully add 40 grams of pasta to the heated water and continue to heat the mixture at
 100 degrees Celsius for a duration of 6 minutes.
```

(c) Experiment 30 - natural language input in writing style 2 (generated using ChatGPT)

```
<XDL>
  <Synthesis>
      <Hardware>
          <Component id="beaker" />
      </Hardware>

      <Reagents>
          <Reagent name="water" />
          <Reagent name="pasta" />
      </Reagents>

      <Procedure>
          <Add vessel="beaker" reagent="water" amount="250 g" />
          <HeatChill vessel="beaker" temp="100" time="0" />
          <Add vessel="beaker" reagent="pasta" amount="40 g" />
          <HeatChill vessel="beaker" temp="100" time="6 min" />
      </Procedure>
  </Synthesis>
</XDL>
```

(d) Experiment 30 - XDL plan generated by CLAIRify for writing style 2

Fig. 17 XDL generated for different writing styles for Experiment 30 from Chem-EDU. The first writing style (writing style 1) is taken directly from Chem-EDU. For the second writing style (writing style 2), we asked ChatGPT to rewrite it. For both prompts, we generate a XDL plan. The output is identical for both

References

Abolhasani, M., & Kumacheva, E. (2023). The rise of self-driving labs in chemical and materials sciences. *Nature Synthesis* 1–10.

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., et al. (2022). Do As I Can, Not As I Say: Grounding language in robotic affordances. arXiv preprint. https://doi.org/10.48550/arXiv.2204.01691 .

Baier, J. A., Bacchus, F., & McIlraith, S. A. (2009). A heuristic search approach to planning with temporally extended preferences. *Artificial Intelligence, 173*(5–6), 593–618.

Beeson, P. & Ames, B. (2015) TRAC-IK: An open-source library for improved solving of generic inverse kinematics. In *2015 IEEE-RAS 15th international conference on humanoid robots (humanoids)*.

Berenson, D., Srinivasa, S., & Kuffner, J. (2011). Task space regions: A framework for pose-constrained manipulation planning. *The International Journal of Robotics Research, 30*(12), 1435–1460. https://doi.org/10.1177/0278364910396389

Boiko, D. A., MacKnight, R., & Gomes, G. (2023). Emergent autonomous scientific research capabilities of large language models. arXiv preprint. https://doi.org/10.48550/arXiv.2304.05332

Bran, A. M., Cox, S., White, A. D., & Schwaller, P. (2023). ChemCrow: Augmenting large-language models with chemistry tools. arXiv preprint. https://doi.org/10.48550/arXiv.2304.05376

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in Neural Information Processing Systems, 33*, 1877–1901.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., et al. (2023). Sparks of artificial general intelligence: Early experiments with gpt-4. arXiv preprint. https://doi.org/10.48550/arXiv.2303.12712

Burger, B., Maffettone, P. M., Gusev, V. V., Aitchison, C. M., Bai, Y., Wang, X., Li, X., Alston, B. M., Li, B., Clowes, R., et al. (2020). A mobile robotic chemist. *Nature, 583*(7815), 237–241.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.(2021). Evaluating large language models trained on code. arXiv preprint. https://doi.org/10.48550/arXiv.2107.03374

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). PaLM: Scaling language modeling with pathways. arXiv preprint. https://doi.org/10.48550/arXiv.2204.02311

Coleman, D., Sucan, I., Chitta, S., & Correll, N. (2014). Reducing the barrier to entry of complex robotic software: A MoveIt! case study. arXiv preprint. https://doi.org/10.48550/arXiv.1404.3785

Dantam, N. T., Kingston, Z. K., Chaudhuri, S., & Kavraki, L. E. (2018). An incremental constraint-based framework for task and motion planning. *The International Journal of Robotics Research, 37*(10), 1134–1151.

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *North American chapter of the association for computational linguistics*.

Ding, Y., Zhang, X., Paxton, C., & Zhang, S. (2023). Task and motion planning with large language models for object rearrangement. arXiv preprint. https://doi.org/10.48550/arXiv.2212.09672

Driess, D., Ha, J. S., & Toussaint, M. (2020). Deep visual reasoning: Learning to predict action sequences for task and motion planning from an initial scene image. arXiv preprint. https://doi.org/10.48550/arXiv.2006.05398

Driess, D., Xia, F., Sajjadi, M. S., Lynch, C., Chowdhery, A., Ichter, B., Wahid, A., Tompson, J., Vuong, Q., Yu, T., et al. (2023). Palm-e: An embodied multimodal language model. arXiv preprint. https://doi.org/10.48550/arXiv.2303.03378

Edwards, C., Lai, T., Ros, K., Honke, G., & Ji, H. (2022). Translation between molecules and natural language. arXiv preprint. https://doi.org/10.48550/arXiv.2204.11817

Eppel, S., Xu, H., Bismuth, M., & Aspuru-Guzik, A. (2020). Computer vision for recognition of materials and vessels in chemistry lab settings and the vector-labpics data set. *ACS Central Science, 6*(10), 1743–1752.

Epps, R. W., Bowen, M. S., Volk, A. A., Abdel-Latif, K., Han, S., Reyes, K. G., Amassian, A., & Abolhasani, M. (2020). Artificial chemist: An autonomous quantum dot synthesis bot. *Advanced Materials, 32*(30), 2001626.

Eysenbach, B., Salakhutdinov, R. R., & Levine, S. (2019). Search on the replay buffer: Bridging planning and reinforcement learning. *Advances in Neural Information Processing Systems*, 32.

Fakhruldeen, H., Pizzuto, G., Glowacki, J., & Cooper, A. I. (2022). ARChemist: Autonomous robotic chemistry system architecture. arXiv preprint. https://doi.org/10.48550/arXiv.2204.13571

Fortman, J. J., & Stubbs, K. M. (1992). Demonstrations with red cabbage indicator. *Journal of Chemical Education, 69*(1), 66.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., & Lozano-Pérez, T. (2021). Integrated task and motion planning. *Annual Review of Control, Robotics, and Autonomous Systems, 4*, 265–293.

Garrett, C. R., Lozano-Pérez, T., & Kaelbling, L. P. (2020). PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the 30th international conference on automated planning and scheduling (ICAPS)*, (pp. 440–448). AAAI Press.

Ghallab, M., Howe, A., Knoblock, C, McDermott, D. Ram, A., Veloso, M., Weld, D., Wilkins, D. (1998). PDDL - The Planning Domain Definition Language. Technical Report CVC TR98003/DCS TR1165. New Haven, CT: Yale Center for Computational Vision and Control.

Grinberg, M. (2018). Flask web development: Developing web applications with python. " O'Reilly Media, Inc.".

Gu, Y., Tinn, R., Cheng, H., Lucas, M., Usuyama, N., Liu, X., Naumann, T., Gao, J., & Poon, H. (2021). Domain-specific language model pretraining for biomedical natural language processing. *ACM Transactions on Computing for Healthcare, 3*(1), 1–23. https://doi.org/10.1145/3458754

Häse, F., Roch, L. M., & Aspuru-Guzik, A. (2019). Next-generation experimentation with self-driving laboratories. *Trends in Chemistry, 1*(3), 282–291.

Helmert, M. (2006). The fast downward planning system. *Journal of Artificial Intelligence Research, 26*, 191–246.

Higgins, K., Ziatdinov, M., Kalinin, S. V., & Ahmadi, M. (2021). High-throughput study of antisolvents on the stability of multicomponent metal halide perovskites through robotics-based synthesis and machine learning approaches. *Journal of the American Chemical Society, 143*(47), 19945–19955.

Huang, D. A., Nair, S., Xu, D., Zhu, Y., Garg, A., Fei-Fei, L., Savarese, S., & Niebles, J. C. (2019). Neural task graphs: Generalizing to unseen tasks from a single video demonstration. In *IEEE Computer Vision and Pattern Recognition*.

Huang, W., Abbeel, P., Pathak, D., & Mordatch, I. (2022). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, (pp. 9118–9147). PMLR.

Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., et al. (2022). Inner monologue: Embodied reasoning through planning with language models. arXiv preprint. https://doi.org/10.48550/arXiv.2207.05608

Huang, Y., Wilches, J., & Sun, Y. (2021). Robot gaining accurate pouring skills through self-supervised learning and generalization. *Robotics and Autonomous Systems, 136*, 103692. https://doi.org/10.1016/j.robot.2020.103692

Inagaki, T., Kato, A., Takahashi, K., Ozaki, H., & Kanda, G. N. (2023). LLMs can generate robotic scripts from goal-oriented instructions in biological laboratory automation. arXiv preprint. https://doi.org/10.48550/arXiv.2304.10267

Irwin, R., Dimitriadis, S., He, J., & Bjerrum, E. J. (2022). Chemformer: A pre-trained transformer for computational chemistry. *Machine Learning: Science and Technology, 3*(1), 015022.

Jablonka, K. M., Schwaller, P., Ortega-Guerrero, A., & Smit, B. (2023). Is gpt-3 all you need for low-data discovery in chemistry? ChemRxiv. https://doi.org/10.26434/chemrxiv-2023-fw8n4

Kaelbling, L. P., & Lozano-Pérez, T. (2011). Hierarchical task and motion planning in the now. In *IEEE International Conference on Robotics and Automation* (pp. 1470–1477). IEEE.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research, 30*(7), 846–894.

Kavraki, L. E., Svestka, P., Latombe, J. C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation, 12*(4), 566–580.

Kennedy, M., Schmeckpeper, K., Thakur, D., Jiang, C., Kumar, V., & Daniilidis, K. (2019). Autonomous precision pouring from unknown containers. *IEEE Robotics and Automation Letters, 4*(3), 2317–2324. https://doi.org/10.1109/LRA.2019.2902075

Khodeir, M., Agro, B., & Shkurti, F. (2023). Learning to search in task and motion planning with streams. *IEEE Robotics and Automation Letters, 8*(4), 1983–1990.

Khodeir, M., Sonwane, A., & Shkurti, F. (2022). Policy-guided lazy search with feedback for task and motion planning. arXiv preprint. https://doi.org/10.48550/arXiv.2210.14055

Kim, B., Shimanuki, L., Kaelbling, L. P., & Lozano-Pérez, T. (2022). Representation, learning, and planning algorithms for geometric task and motion planning. *The International Journal of Robotics Research, 41*(2), 210–231.

Kingston, Z., Moll, M., & Kavraki, L. E. (2018). Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems, 1*, 159–185.

Kingston, Z., Moll, M., & Kavraki, L. E. (2019). Exploring implicit spaces for constrained sampling-based planning. *The International Journal of Robotics Research, 38*(10–11), 1151–1178. https://doi.org/10.1177/0278364919868530

Kitchener, B. G., Wainwright, J., & Parsons, A. J. (2017). A review of the principles of turbidity measurement. *Progress in Physical Geography, 41*(5), 620–642.

Knobbe, D., Zwirnmann, H., Eckhoff, M., & Haddadin, S. (2022). Core processes in intelligent robotic lab assistants: Flexible liquid handling. In *2022 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 2335–2342.

Le, H., Wang, Y., Gotmare, A. D., Savarese, S., & Hoi, S. C. H. (2022). CodeRL: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems, 35*, 21314–21328.

Li, J., Li, J., Liu, R., Tu, Y., Li, Y., Cheng, J., He, T., & Zhu, X. (2020). Autonomous discovery of optically active chiral inorganic perovskite nanocrystals through an intelligent cloud lab. *Nature Communications, 11*(1), 2046.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. (2022). Competition-level code generation with alphacode. *Science, 378*(6624), 1092–1097.

Liang, J., Huang, W., Xia, F., Xu, P., Hausman, K., Ichter, B., Florence, P., & Zeng, A. (2022). Code as policies: Language model programs for embodied control. arXiv preprint. https://doi.org/10.48550/arXiv.2209.07753

Lim, J. X. Y., Leow, D., Pham, Q. C., & Tan, C. H. (2020). Development of a robotic system for automatic organic chemistry synthesis. *EEE Transactions on Automation Science and Engineering, 18*(4), 2185–2190.

Lin, K., Agia, C., Migimatsu, T., Pavone, M., & Bohg, J. (2023). Text2Motion: From natural language instructions to feasible plans. arXiv preprint. https://doi.org/10.48550/arXiv.2303.12153

Liu, R., Wei, J., Gu, S.S., Wu, T.Y., Vosoughi, S., Cui, C., Zhou, D., & Dai, A.M. (2023). Mind's eye: Grounded language model reasoning through simulation. In *The eleventh international conference on learning representations*.

Macarron, R., Banks, M. N., Bojanic, D., Burns, D. J., Cirovic, D. A., Garyantes, T., Green, D. V., Hertzberg, R. P., Janzen, W. P., Paslay, J. W., et al. (2011). Impact of high-throughput screening in biomedical research. *Nature Reviews Drug discovery, 10*(3), 188–195.

MacLeod, B. P., Parlane, F. G., Morrissey, T. D., Häse, F., Roch, L. M., Dettelbach, K. E., Moreira, R., Yunker, L. P., Rooney, M. B., Deeth, J. R., et al. (2020). Self-driving laboratory for accelerated discovery of thin-film materials. *Science Advances, 6*(20), eaaz8867.

Mehr, H., Craven, M., Leonov, A., Keenan, G., & Cronin, L. (2020a). Benchmarking results and the XDL XML schema. https://zenodo.org/record/3955107

Mehr, S. H. M., Craven, M., Leonov, A. I., Keenan, G., & Cronin, L. (2020). A universal system for digitization and automatic execution of the chemical synthesis literature. *Science, 370*(6512), 101–108.

Ménard, A. D., & Trant, J. F. (2020). A review and critique of academic lab safety research. *Nature Chemistry, 12*(1), 17–25.

Mialon, G., Dessì, R., Lomeli, M., Nalmpantis, C., Pasunuru, R., Raileanu, R., Rozière, B., Schick, T., Dwivedi-Yu, J., Celikyilmaz, A., et al. (2023). Augmented language models: a survey. arXiv preprint. https://doi.org/10.48550/arXiv.2302.07842

Mirchandani, S., Karamcheti, S., & Sadigh, D. (2021). ELLA: Exploration through learned language abstraction. *Advances in Neural Information Processing Systems, 34*, 29529–29540.

Mishra, S., Khashabi, D., Baral, C., & Hajishirzi, H. (2021). Cross-task generalization via natural language crowdsourcing instructions. arXiv preprint. https://doi.org/10.48550/arXiv.2104.08773

Muchacho, R. I. C., Laha, R., Figueredo, L. F., & Haddadin, S. (2022). A solution to slosh-free robot trajectory optimization. In *2022 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, (pp. 223–230). IEEE.

National Astronomical Observatory of Japan. (2022). *Handbook of scientific tables*. World Scientific.

Ni, A., Iyer, S., Radev, D., Stoyanov, V., Yih, W. T., Wang, S. I., & Lin, X. V. (2023). Lever: Learning to verify language-to-code generation with execution. arXiv preprint. https://doi.org/10.48550/arXiv.2302.08468

Olson, E. (2011). Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE the international conference on robotics and automation*.

Peng, B., Galley, M., He, P., Cheng, H., Xie, Y., Hu, Y., Huang, Q., Liden, L., Yu, Z., Chen, W., & Gao, J. (2023). Check your facts and try again: Improving large language models with external knowledge and automated feedback. arXiv preprint. https://doi.org/10.48550/arXiv.2302.12813

Pereira, D., & Williams, J. (2007). Origin and evolution of high throughput screening. *British Journal of Pharmacology, 152*(1), 53–61.

Perry, T. (2021). LightTag: Text annotation platform. In *Proceedings of the EMNLP conference*, (pp. 20–27).

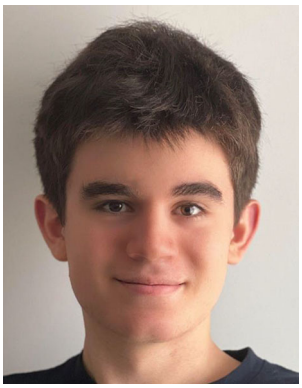
Pizzuto, G., Wang, H., Fakhruldeen, H., Peng, B., Luck, K. S., & Cooper, A.I . (2022). Accelerating laboratory automation through robot

skill learning for sample scraping. arXiv preprint. https://doi.org/10.48550/arXiv.2209.14875
Ramos, M. C., Michtavy, S. S., Porosoff, M. D., & White, A. D. (2023). Bayesian optimization of catalysts with in-context learning. arXiv preprint. https://doi.org/10.48550/arXiv.2304.05341
Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. arXiv preprint. https://doi.org/10.48550/arXiv.2302.04761
Seifrid, M., Pollice, R., Aguilar-Granda, A., Morgan Chan, Z., Hotta, K., Ser, C. T., Vestfrid, J., Wu, T. C., & Aspuru-Guzik, A. (2022). Autonomous chemical experiments: Challenges and perspectives on establishing a self-driving lab. *Accounts of Chemical Research, 55*(17), 2454–2466.
Shah, D., Xu, P., Lu, Y., Xiao, T., Toshev, A., Levine, S., & Ichter, B. (2021). Value function spaces: Skill-centric state abstractions for long-horizon reasoning. arXiv preprint. https://doi.org/10.48550/arXiv.2111.03189
Sharma, P., Torralba, A., & Andreas, J. (2021). Skill induction and planning with latent language. arXiv preprint. https://doi.org/10.48550/arXiv.2110.01517
Shiri, P., Lai, V., Zepel, T., Griffin, D., Reifman, J., Clark, S., Grunert, S., Yunker, L. P., Steiner, S., Situ, H., et al. (2021). Automated solubility screening platform using computer vision. *iscience, 24*(3), 102176.
Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., Fox, D., Thomason, J., & Garg, A. (2022). Progprompt: Generating situated robot task plans using large language models. arXiv preprint. https://doi.org/10.48550/arXiv.2209.11302
Steiner, S., Wolf, J., Glatzel, S., Andreou, A., Granda, J. M., Keenan, G., Hinkley, T., Aragon-Camarasa, G., Kitson, P. J., Angelone, D., et al. (2019). Organic synthesis in a modular robotic system driven by a chemical programming language. *Science, 363*(6423), eaav2211.
Taylor, R., Kardas, M., Cucurull, G., Scialom, T., Hartshorn, A., Saravia, E., Poulton, A., Kerkez, V., & Stojnic, R. (2022). Galactica: A large language model for science. arXiv preprint. https://doi.org/10.48550/arXiv.2211.09085
Tellex, S., Kollar, T., Dickerson, S., Walter, M., Banerjee, A., Teller, S., & Roy, N. (2011). Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the AAAI conference on artificial intelligence*, vol. *25*, pp. 1507–1514.
Toussaint, M. (2015). Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pp. 1930–1936.
Toussaint, M. A., Allen, K. R., Smith, K. A., & Tenenbaum, J. B. (2018). Differentiable physics and stable modes for tool-use and manipulation planning.
Wang, S., Liu, Y., Xu, Y., Zhu, C., & Zeng, M. (2021). Want to reduce labeling cost? GPT-3 can help. In *Proceedings of the EMNLP Conference*, pp. 4195–4205.
Wang, Y., Wang, W., Joty, S., & Hoi, S. C. (2021). CodeT5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. arXiv preprint. https://doi.org/10.48550/arXiv.2109.00859
Wang, Y. R., Zhao, Y., Xu, H., Eppel, S., Aspuru-Guzik, A., Shkurti, F., & Garg, A. (2023). MVTrans: Multi-view perception of transparent objects. arXiv preprint. https://doi.org/10.48550/arXiv.2302.11683
Wolthuis, E., Pruiksma, A. B., & Heerema, R. P. (1960). Determination of solubility: A laboratory experiment. *Journal of Chemical Education, 37*(3), 137.
Wu, C. J., Raghavendra, R., Gupta, U., Acun, B., Ardalani, N., Maeng, K., Chang, G., Aga, F., Huang, J., Bai, C., et al. (2022). Sustainable AI: Environmental implications, challenges and opportunities. *Proceedings of Machine Learning and Systems, 4*, 795–813.
Xu, D., Martín-Martín, R., Huang, D. A., Zhu, Y., Savarese, S., & Fei-Fei, L. F. (2019). Regression planning networks. *Advances in Neural Information Processing Systems*, 32.
Xu, D., Nair, S., Zhu, Y., Gao, J., Garg, A., Fei-Fei, L., & Savarese, S. (2018). Neural task programming: Learning to generalize across hierarchical tasks. In *2018 IEEE international conference on robotics and automation (ICRA)* (pp. 3795–3802). IEEE.
Xu, H., Wang, Y. R., Eppel, S., Aspuru-Guzik, A., Shkurti, F., & Garg, A. (2021). Seeing glass: Joint point-cloud and depth completion for transparent objects. In *Annual conference on robot learning*.
Yoshikawa, N., Darvish, K., Garg, A., & Aspuru-Guzik, A. (2023). Digital pipette: Open hardware for liquid transfer in self-driving laboratories. Digital Discovery. https://doi.org/10.1039/d3dd00115f
Yoshikawa, N., Li, A. Z., Darvish, K., Zhao, Y., Xu, H., Kuramshin, A., Aspuru-Guzik, A., Garg, A., & Shkurti, F. (2023). Chemistry lab automation via constrained task and motion planning. arXiv preprint. https://doi.org/10.48550/arXiv.2212.09672
Yoshikawa, T. (1985). Manipulability of robotic mechanisms. *The International Journal of Robotics Research, 4*(2), 3–9. https://doi.org/10.1177/027836498500400201
Zhang, K., Sharma, M., Liang, J., & Kroemer, O. (2020). A modular robotic arm control stack for research: Franka-Interface and FrankaPy. arXiv preprint. https://doi.org/10.48550/arXiv.2011.02398

Naruki Yoshikawa is a Ph.D. student at the Department of Computer Science of the University of Toronto. He is working on automation of chemistry experiments under the supervision of Alán Aspuru-Guzik. He received his master's degree in 2020 at The University of Tokyo and his bachelor's degree in 2018 at the same institution.

Marta Skreta is a Computer Science PhD student under the supervision of Alán Aspuru-Guzik at the University of Toronto working at the intersection of machine learning, chemistry, and self-driving labs. Previously, she completed a MSc in Computer Science at the University of Toronto and a HBSc in Chemical Biology at McMaster University.

Kourosh Darvish is a postdoctoral researcher at the Computer Science and Robotics Institute of the University of Toronto (UofT) and a member of the Vector Institute. Before joining UofT in 2022, he was a postdoctoral researcher at the Italian Institute of Technology (IIT). In 2019, he completed his PhD in Bioengineering and Robotics from the University of Genoa, Italy. Previously, he received his B.Sc. and M.Sc. degrees in Aerospace Engineering from K.N. Toosi University of Technology and Sharif University of Technology (Tehran, Iran) in 2012 and 2014, respectively. His research focuses on robotics for scientific discoveries, shared autonomy, and humanoid robotics.



Sebastian Arellano-Rubach is in grade 11 at the University of Toronto Schools. In 2023, he spent the summer working for the Computer Science Department at UofT. His work focuses mostly on website design, front-end development, back-end development, development tools, performance optimization, responsiveness, and SEO.

Zhi Ji is a dedicated Engineering Science student at the University of Toronto, specializing in Machine Intelligence (AI) and minoring in Engineering Business. She is very passionate about AIMLDL.

Lasse Bjørn Kristensen is a postdoctoral researcher working in Alán Aspuru-Guzik's lab on quantum computing, machine learning, and algorithms at the intersection of the two. He obtained his PhD from Aarhus University in 2020, where his research focus was on the use of superconducting circuits and their quantum dynamics for quantum computing. Before this, he received his MSc in 2017 and his BSc in 2015 at the same institution, both within the field of Physics.

Andrew Zou Li is a senior undergraduate student studying robotics in the Division of Engineering Science at the University of Toronto. His research experience includes state estimation for autonomous driving, task and motion planning, and machine learning compiler optimization.

Yuchi Zhao is a research assistant in the Computer Science Department of the University of Toronto. His research mainly focuses on robot manipulation, perception and tactile sensing for self-driving labs. He received his B.A.Sc degree in Mechatronics Engineering from the University of Waterloo in 2023.

Haoping Xu is a senior Ph.D. student from RA^{2D}: Robotics-assisted Accelerated Discovery lab at the Acceleration Consortium. He is supervised by Prof Alán Asupuru-Guizk, Prof Animesh Grag and Prof Florian Shkurti. RA^2D lab is focused on building universal robot for lab automation and accelerating science discoveries. Haoping's research focus is computer vision and robotics control related to lab automation. In particular, he is interested in perception and vision guided robot planning and control for transparent objects. He has publications in topics of 2D perception, depth completion, multiview 3D perception and robot manipulations for transparent objects in lab settings.

Artur Kuramshin recieved the HBSc degree in Computer Science with a specialization in Computer Vision from the University of Toronto (UofT) in 2023. During his time at UofT, he was part of the Robot Vision and Learning (RVL) lab.

Alán Aspuru-Guzik research lies at the interface of computer science with chemistry and physics. He integrates robotics, machine learning and quantum chemistry to develop "self-driving laboratories", accelerating rates of scientific discovery. He develops quantum computer algorithms and has pioneered quantum algorithms for the simulation of matter. He is a Professor of Chemistry and Computer Science at the University of Toronto, faculty member at the Vector Institute for Artificial Intelligence and Director of the Acceleration Consortium, a University of Toronto-based strategic initiative that aims to gather researchers from industry, government and academia around topics related to the labs of the future. He was previously a full professor at Harvard University where he started his career in 2006. He is currently the Canada 150 Research Chair in Theoretical Chemistry, CIFAR AI Chair at the Vector Institute and co-founder of Zapata Computing and Kebotix, two early-stage ventures in quantum computing and self-driving laboratories.

Florian Shkurti is an Assistant Professor in the Department of Computer Science at the University of Toronto, where he also serves as a Faculty Member at the UofT Robotics Institute and a Faculty Affiliate at the Vector Institute. He earned his Ph.D. in Computer Science and Robotics from McGill University in 2018. He leads the Robot Vision and Learning (RVL) lab, focusing on robotics research encompassing machine learning, perception, planning, and control. Florian's research has been applied to environmental monitoring by autonomous robots, visual navigation for autonomous vehicles, and mobile manipulation.

Animesh Garg received the PhD degree from University of California, Berkeley and was a postdoc at Stanford AI lab. He is a CIFAR chair Assistant Professor of computer science with the University of Toronto, a faculty member with the Vector Institute, and a senior research scientist with Nvidia. He works on the Algorithmic Foundations for Generalizable Autonomy, to enable AI-based robots to work alongside humans.