HIERARCHICALLY ENCAPSULATED REPRESENTATION FOR PROTOCOL DESIGN IN SELF-DRIVING LABS

Yu-Zhe Shi[1 ⋆], **Mingchen Liu**[2 ⋆], **Fanxu Meng**[1], **Qiao Xu**[1], **Zhangqian Bi**[2], **Kun He**[2], **Lecheng Ruan**[1 ✉], **Qining Wang**[1 ✉]

[1] Department of Advanced Manufacturing and Robotics, Peking University

[2] School of Computer Science and Technology, Huazhong University of Science and Technology

⋆Equal contribution ✉ ruanlecheng@ucla.edu, qiningwang@pku.edu.cn

ABSTRACT

Self-driving laboratories have begun to replace human experimenters in performing single experimental skills or predetermined experimental protocols. However, as the pace of idea iteration in scientific research has been intensified by Artificial Intelligence, the demand for rapid design of new protocols for new discoveries become evident. Efforts to automate protocol design have been initiated, but the capabilities of knowledge-based machine designers, such as Large Language Models, have not been fully elicited, probably for the absence of a systematic representation of experimental knowledge, as opposed to isolated, flatten pieces of information. To tackle this issue, we propose a multi-faceted, multi-scale representation, where instance actions, generalized operations, and product flow models are hierarchically encapsulated using Domain-Specific Languages. We further develop a data-driven algorithm based on non-parametric modeling that autonomously customizes these representations for specific domains. The proposed representation is equipped with various machine designers to manage protocol design tasks, including planning, modification, and adjustment. The results demonstrate that the proposed method could effectively complement Large Language Models in the protocol design process, serving as an auxiliary module in the realm of machine-assisted scientific exploration.

1 INTRODUCTION

The rapid advancement of Artificial Intelligence (AI) models for the assistance of scientific discovery (Wang et al., 2023b) has precipitated an increased demand for rapid iteration of ideas, from the generation to the verification of hypotheses. Although AI models have expedited the process of hypothesis generation, the validation phase still requires intensive empirical experimentation from human. The concept of self-driving laboratory has been introduced to substantially accelerate the validation process, in organic chemical synthesis (Mehr et al., 2020; Burger et al., 2020), cell biology for medical research (Kanda et al., 2022), and novel material discovery (Szymanski et al., 2023). With the expertise and effort of experimental scientists and automation engineers, mobile robots and Internet of Things (IoT) pipelines are configured to perform a sequence of actions in accordance with a detailed description of the specific experimental procedure, referred to as the *protocol*.

While existing protocols suffice for some experimental tasks, discovery processes often demand a higher degree of specificity, including: (i) confirmation of unverified experimental objectives to seek specific findings; (ii) testing parallel hypotheses or solutions; and (iii) replication of established experiments within the constraints of available laboratory resources. These necessitate the *design* of new protocols, going beyond the reuse of existing ones available in the protocol databases. Particularly, this includes the *planning* of novel protocols, and the *modification* and *adjustment* of current protocols as appropriate, respectively. Unfortunately, self-driving laboratories currently only execute isolated and duplicated experimental skills (Bédard et al., 2018; Steiner et al., 2019), or pre-specified protocols with sequential actions (Rohrbach et al., 2022; Manzano et al., 2022). Any innovation in protocols imposes intensive manual design burden (McNutt, 2014; Baker, 2016), potentially becoming a bottleneck in accelerating scientific discovery. Consequently, there is a quest for the automatic design of protocols tailored to specific goals for self-driving laboratories.

Designing new protocols is a non-trivial task even for human scientists. Novice scientists tend to adhere strictly to established protocols and may be at a loss when faced with the need for variations, from minor adjustments like different available devices to more significant shifts in the overall experimental goal. In contrast, veteran scientists typically have the capability to create or modify protocols as needed, from variations in available resources (*"what I have"*) to desired outcomes (*"what I want"*), even in situations where a similar protocol was not encountered before.

The distinction arises because veteran scientists possess a *systematic* understanding of every ingredient and procedure, contextualizing them globally within the domain of experiment. They know *"what kind of ingredient is used for what purposes"* and *"what kind of operation is used under what conditions'*, while novice scientists mechanically memorize the sequential execution orders and corresponding parameters in a local context. This systematic understanding, or *conceptual knowledge* (Ryle & Tanney, 1949), includes the background knowledge of ingredients and atomic operations, as well as the relationships between them. Experienced experimental scientists develop such conceptual knowledge as a *representation* for protocol design (McCarthy, 1959), which serves as the vehicle for reasoning processes. Reasoning over conceptual knowledge leverages the rich context of generalized, abstracted concepts of ingredients and operations rather than specified, instantialized ones, which spans a semantic space where originally isolated dots are connected with each other, thereby enhancing the simplicity and flexibility of protocol design (Boden, 1980; Newell, 1982). In summary, veteran scientists' capability to design new protocols stems from an appropriate representation of background knowledge that supports reasoning processes (see Fig. 1A).

To implement automatic protocol design on machines, a reasonable choice may be leveraging a Large Language Model (LLM). Trained on extensive corpora, including scientific documents, LLMs possess the potential to facilitate protocol design with the corresponding background knowledge (AI4Science & Quantum, 2023). Recently, researchers have made beneficial attempts to design new protocols using LLMs based on descriptions of new experimental goals (Boiko et al., 2023; M. Bran et al., 2024). Regrettably, benchmarking results indicate that the expected capability of LLMs in protocol design is not fully elicited (O'Donoghue et al., 2023). One significant limitation is that LLMs excel at generating new protocols similar to existing ones, *i.e.*, protocols with similar sequential execution orders, but fail to generate those with distinct dependency distributions. This limitation hampers LLMs in scenarios where experimental goals change in high intensity. Another limitation is that the generated protocols sometimes lose critical configuration details for operation execution, necessitating manual correction. These empirical evidences suggest that LLMs exhibit limitations akin to those of novice human experts, implying that LLMs may necessitate a more suitable representation of background knowledge to fully unleash their potential in protocol design.

Protocol design is a multi-faceted, multi-scale effort requiring the integration of information from different perspectives, from low-level to high-level. This information includes detailed configurations of each atomic operation, temporal relationships between atomic operations, the scope of application for atomic operations with the same reference name, and the reactive relationships between reagents and operations. While LLMs undoubtedly capture such knowledge from their training corpora, the pieces of knowledge remain isolated, unorganized, and not articulated. These flatten background knowledge, rather than conceptual knowledge, hinders LLMs from *flying over* a global view of the novel objectives and *diving into* the details of operations. Therefore, we propose developing a **multi-faceted and multi-scale representation** for protocol design that provides the designer, such as LLMs, with a vehicle to reason over conceptual knowledge of ingredients and procedures.

We draw inspiration from both cognitive science literature on rationality (Monsell, 2003; Griffiths, 2020), which suggests that we cannot consider information from different views and scales in a single thread (Shi et al., 2023a). We also learn from computer science literature on hierarchical abstraction (Liskov, 1987), which indicates that higher-level abstraction semantics possess more powerful expressivity compared to their lower-level counterparts (Abelson & Sussman, 1996; Hopcroft et al., 1996). Combining these insights, we suggest that our desired representation should encapsulate information of different granularities in corresponding hierarchies of abstraction, gaining global design insights with higher-level semantics while completing execution configurations with lower-level semantics. Specifically, we investigate three levels of encapsulation (see Fig. 3B). Starting from the set of original protocols, namely the basic level, we have (i) **protocol element instantialization**, which decomposes full protocols into instance operations with attributes, within the local context of the specific protocol, resulting a structural representation of the elementary information; (ii) **function abstraction**, which offers an operation-centric view that generalizes the precondition, postcondition, and execution configurations of each operation in the global context of the experiment domain, resulting a sequential representation of the operations; (iii) **model abstraction**, which offers an reagent and intermediate product centric view that unifies the status transitions in the global context of the experiment domain, resulting a continuous representation of the experimental environ-

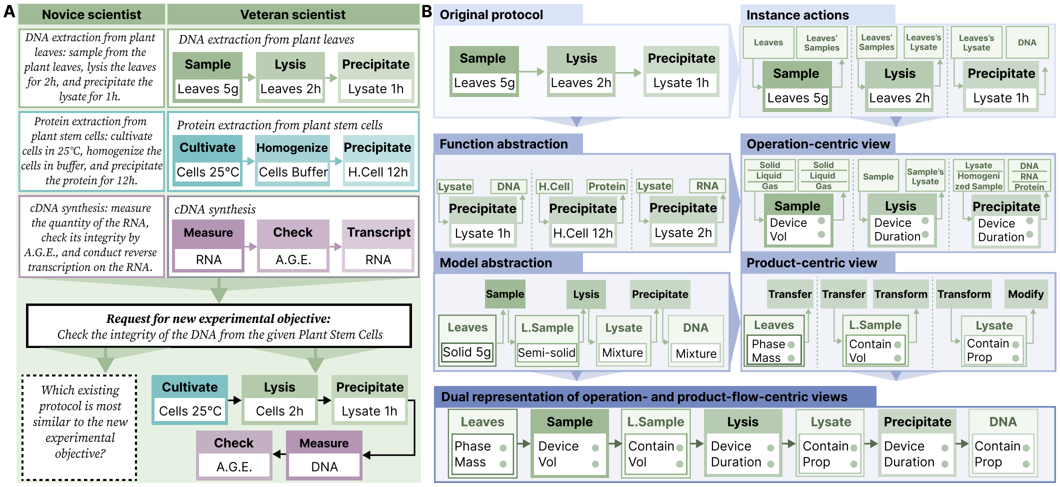


Figure 1: **The representations for protocol design.** **(A)** The example of protocol design by novice and veteran experimental scientists. **(B)** The hierarchies of our proposed representation, from original full protocol representation, to dual representation of operation- and product-flow-centric views.

ment. This hierarchical structure provides the designer with a representation to consider all possible associations among operations, among products, and between operations and products, with a high degree of freedom, by disentangling originally intertwined information. We implement the representation using Domain-Specific Languages (DSLs). The hierarchical syntax of DSLs maintains both the abstract semantics at the high-level and the precise information at the low-level. Furthermore, the *compositionality* of DSL syntax facilitates the flexible protocol designs, addressing the "flying over global views" requirement; while DSL program verification over the generated protocols upholds their *soundness* and *completeness*; addressing the "diving into details" requirements.

However, the proposed representation does not come without drawbacks—it can be highly dependent on domain-specific knowledge (Mernik et al., 2005; Fowler, 2010). The distributions of reagents, operations, and execution dependencies vary significantly across different domains in experimental sciences, such as *Genetics*, *Medical*, *Bioengineering*, and *Ecology*. Manually crafting DSLs specialized for these domains requires deep integration between domain experts and programming language experts, which is labour-intensive, case-by-case, and costly (Shi et al., 2024a;b). This obstacle hinders the application of our representation to a broader set of domains (Shi et al., 2024d). To make the representation specification more affordable, we develop an algorithm that conducts multi-hierarchy encapsulation automatically driven by the domain-specific corpus of existing protocols. Ultimately, we may be able to take a critical step toward closing the loop of *autonomous* scientific discovery by establishing these two building blocks: (i) the automatic generation of **representation** for protocol design; and (ii) the automatic **designer** working on the representation.

Our contributions in this work are three-fold: (i) we identify the problem of representation for protocol design and develop a hierarchically encapsulated representation for protocol design (Sec. 2); (ii) we propose a data-driven algorithm that automatically generates the representation for protocol design specialized for the domain of application (Sec. 3); and (iii) we demonstrate the utility of the resulting representation by conducting protocol planning, modification and adjustment tasks using a variety of machine designers across different domains (Sec. 4). This further indicates that our proposed automatic representation generation approach possesses the potential to function as an auxiliary module for LLMs, enhancing their capability on protocol design.

2 Representation for protocol design

In this section, we describe our representation for protocol design (see Fig. 1B). We first formulate the basic protocol design problem in Sec. 2.1. Afterwards, starting from the original full protocol, we introduce the three hierarchies of representations: (i) structural representation, *i.e.*, instance actions with attributes (Sec. 2.2); (ii) sequential operation-centric representation, *i.e.*, function abstraction (Sec. 2.3); and (iii) continuous product-flow-centric representation, *i.e.*, model abstraction (Sec. 2.4). Furthermore, we describe how the dual representation of operation-centric and product-flow-centric views reciprocatively facilitates the verification of the designed protocols in Sec. 2.5.

2.1 The protocol design problem

Protocol design problem PD $= (\Phi \mid \omega^*, \mathcal{P}, \Omega)$ is generating a desired protocol Φ given the **new coming** experimental objective ρ, domain of experiment $\mathcal{P}$, and available reagents Ω. A protocol $\Phi = \langle \varphi_1, \varphi_2, \dots \rangle$ is a sequence of experimental steps φ_t. An experimental objective ω^* is the expected final product of the experiment. Experimental objectives can range from preparing a desired product, to testing the significance of a specific hypothesis and detecting a predicted behavior, with the latter two potentially followed by additional standalone steps for property test, observation, and interpretation. We denote domains of experiment as $\mathcal{P}$, which influences the distributions of protocols by means of the distributions of operations, reagents, and execution orders, *etc*. The set of available reagents Ω includes originally accessible reagents and excludes those requiring production.

2.2 Instance actions with attributes

Protocols are originally represented in Natural Language (NL), which is the representation suitable for humans' comprehension, but not for machines (Bartley et al., 2023). Without a syntax decomposing a NL-based protocol into information elements precisely, machines are likely to capture only overall, coarse-grained information of protocols and may only retrieve within existing protocols for the one that is most similar to the new experimental objective. Consequently, according to the standards and conventions of experimental sciences (Baker, 2021), the prerequisite of representation for a machine protocol designer should be a structural representation which decompose NL-based protocols into instance actions with attributes $\{\varphi_t \mid (\varphi_t^{\text{prec}}, \varphi_t^{\text{post}}, \varphi_t^{\text{exec}})\}$. The instance actions are decomposed by execution order and their attributes are the exact context for their execution, namely the precondition φ_t^{prec}, *i.e.*, the availability of resources required for this action, postcondition φ_t^{post}, *i.e.*, resulting product of the operation, and execution configurations φ_t^{exec}. Execution configurations includes the configuration parameters and their corresponding values, *e.g.*, the device for conducting the operation and required experimental conditions such as duration, acidity, and lightening. An instance action can be reusable in another protocol once the execution context is matched.

With such reusability, we are on the first time to have *building blocks* for constructing a new protocol rather than retrieving existing ones. These building blocks capture fine-grained execution configuration parameters through maintaining the nested data structures of key-value pairs. This structural representation serves as a syntactic constraint on the preciseness of designed protocols. Practical attempts have been made echoing this idea (O'Donoghue et al., 2023; Leonov et al., 2024).

2.3 Operation-centric view with function abstraction

The reusability of instance actions with attributes is highly limited, as their semantics are highly specified in the low-level. The total amount of the instance actions can be extremely high, *i.e.*, about 150K per domain, thus the probability of the exact matching between execution contexts can be extremely low. Consider the three different instance actions with attributes *"Homogenization of mouse liver tissue using a bead mill"*, *"Homogenization of bacterial cell suspension using an ultrasonic homogenizer"*, and *"Homogenization of bacterial air samples using a nebulizer"*. Although they come with totally different preconditions, postconditions, and execution configurations, particularly the required device varying according to the phase of the experimental subject, they share the semantic identifier *"Homogenization"* for reference. Sharing semantic identifier indicates that these instance actions share the same *purpose* on the semantics level. In experimental sciences, *"Homogenization"* always refers to the breakdown of a sample into a uniform mixture. Whether it's tissue, cell suspension, or gas doesn't change the purpose of the operation. This is critical for protocol design, since it essentially requires satisfying the ultimate goal through a series of subgoals. Therefore, the desired representation should generalize the semantics of operations to any possible contexts in the corresponding domain of experiment, rather than only specific contexts.

We implement such generalization by encapsulating varied instances of preconditions, postconditions, and execution configurations into an *interface* for the operation. Namely, we refer to an operation with semantic identifier φ through an interface ϕ to a set of execution contexts, in the form of $\langle \varphi \mapsto \phi \mapsto \{(\varphi^{\text{prec}}, \varphi^{\text{post}}, \varphi^{\text{exec}})\}\rangle$. The operation φ can be grounded to a corresponding instance action in any matched execution contexts, echoing *modular design* (Hirtz et al., 2002). The reusability of encapsulated operations comes with greater significance than that of instance actions, as there are only about 1K operations per domain in total, which is only $1/150$ of that of instance actions. As flexible building blocks, operations can be easily fitted into any breakpoints with suitable preconditions and postconditions in the constructing experiment sequence. This sequential representation of the operations serves as a semantic constraint on the compact permissible set of primitives for protocol design (Shi et al., 2023b), maintaining both degree of freedom and correctness.

2.4 Product-flow-centric view with model abstraction

Sequence of operations make up of protocols. However, operations are the methods to realize rather than the objectives to achieve. For experimental objectives of testing, preparing, or detecting (Schwab & Held, 2020), the common focus is always the specific status of final product, not the operations. Starting from initial reagents, the status of product flow is manipulated step-by-step by the operations, till the final product. Unfortunately, the information of product status transition is *latent* in protocols and is *twisted with* descriptions of experimental steps. For the operation-centric view, the transitions of product flow statuses remains a *black box environment*. For example, the operation description *"Centrifuge the tubes at 15,000 x g for 20 minutes"* does not directly reveal the transition from product in mixture status to products in distinct phases. The lack of coherent tracking of the product flow is problematic of protocol design, as the product flow holds *spatial-temporal invariance*, just the same as the general physical environment. Status transitions of the product flow are primarily caused by the effects of operations, thereby it serves as the *invariant* in executing the protocol from the perspective of programming. Therefore, the desired representation should also serve as the *model* interacting with the sequence of operations.

To disentangle product status from their latent representation in the operation-centric view, we propose an explicit product flow centric view that tracks the status of the product flows with detail, such as component, volume, container, and other physical and chemical properties of the product, and also the predecessor operation that yields the product and the successor operation that takes the product as input. Each product flow unit, *i.e.*, one individual component in the product flow between two adjacent steps, is an instance with attributes $\{\omega_t \mid (\omega_t^{\text{pred}}, \omega_t^{\text{succ}}, \omega_t^{\text{prop}})\}$. Analogous to the generalization of operations' semantics, product flow units share commonalities between components with the same semantic identifier for reference—they may share a specific range of predecessor operations ω^{pred} and successor operations ω^{succ}, and a selected set of key properties to consider ω^{prop}. For example, the *"supernatant"* is usually generated by a *"centrifugation"* operation, passing into *"filtration"* or *"spectrophotometric analysis"*, and focusing on the properties *acidity* and *viscosity* rather than other possible properties. Thus, we encapsulate the information of contexts and properties into the semantics of product flow units, in the form of $\langle \omega \mapsto (\omega^{\text{pred}}, \omega^{\text{succ}}, \omega^{\text{prop}}) \rangle$. As solid pipelines bridging the building blocks, product flow units can verify the coherency of the entire designed protocol. This continuous representation of the environments serves as a program verifier, checking the prerequisite and simulating the effect of each operation, alleviating unpredictable behaviors among the interaction between operations and product flows.

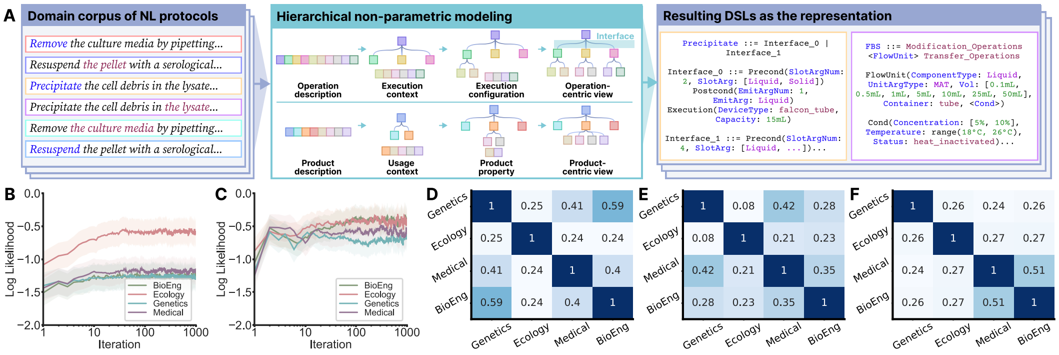
2.5 Reciprocative verification over the dual representation

The dual representation of operation-centric and product-flow-centric views intrinsically equips with a verification mechanism through a reciprocative process akin to two interacting threads. The first thread focuses on verifying the operation flow, taking as input an operation φ_t along with its precondition φ_t^{prec} and postcondition φ_t^{post}. The second thread handles the verification of the product flow, taking as input a product ω_t along with its predecessor operation ω_t^{pred} and successor operation ω_t^{succ}.

Algorithm 1 Reciprocative Verification

```
procedure OFVERIFICATION($M$, $\varphi$)
    ▷ Check that the pre/ post conditions are met
    CHECKOPCONDITIONS($\varphi, \varphi^{\text{prec}}, \varphi^{\text{post}}$)
    if $\varphi^{\text{prec}} \subseteq M(\Omega)$ then
        $M(\Omega) \leftarrow (M(\Omega) \setminus \varphi^{\text{prec}}) \cup \varphi^{\text{post}}$
        ▷ Proceed to verify each output product
        for product $\omega$ in $\varphi^{\text{post}}$ do
            PFVERIFICATION($M$, $\omega$)
procedure PFVERIFICATION($M$, $\omega$)
    ▷ Check necessary properties of the product
    CHECKPROPERTIES($\omega, \omega^{\text{prop}}, \varphi'^{\text{req}}$)
    ▷ if required by subsequent operations.
    if $\exists\, \varphi'\ s.t.\ \omega \in \varphi'^{\text{prec}}$ then
        ▷ Verify operations using the product
        OFVERIFICATION($M$, $\varphi'$)
```

Specifically, for the operation verification (corresponding to OFVerification in Alg. 1), we ensure that each operation can be correctly executed given its input reagents and that it yields the expected output products. This involves checking that the preconditions are satisfied by the available products from preceding operations and that the postconditions are well-defined for subsequent use. Concurrently, the product flow verification (corresponding to PFVerification in Alg. 1) involves tracking each



Remove the culture media by pipetting...
Resuspend the pellet with a serological...
Precipitate the cell debris in the lysate...
Precipitate the cell debris in the lysate...
Remove the culture media by pipetting...
Resuspend the pellet with a serological...

```
Precipitate ::= Interface_0 |
Interface_1

Interface_0 ::= Precond(SlotArgNum:
2, SlotArg: [Liquid, Solid])
Postcond(EmitArgNum: 1,
EmitArg: Liquid)
Execution(DeviceType: falcon_tube,
Capacity: 15mL)

Interface_1 ::= Precond(SlotArgNum:
4, SlotArg: [Liquid, ...])...
```

```
FBS ::= Modification_Operations
<FlowUnit> Transfer_Operations

FlowUnit(ComponentType: Liquid,
UnitArgType: MAT, Vol: [0.1mL,
0.5mL, 1mL, 5mL, 10mL, 25mL, 50mL],
Container: tube, <Cond>)

Cond(Concentration: [5%, 10%],
Temperature: range(18°C, 26°C),
Status: heat_inactivated)...
```

D

	Genetics	Ecology	Medical	BioEng
Genetics	1	0.25	0.41	0.59
Ecology	0.25	1	0.24	0.24
Medical	0.41	0.24	1	0.4
BioEng	0.59	0.24	0.4	1

E

	Genetics	Ecology	Medical	BioEng
Genetics	1	0.08	0.42	0.28
Ecology	0.08	1	0.21	0.23
Medical	0.42	0.21	1	0.35
BioEng	0.28	0.23	0.35	1

F

	Genetics	Ecology	Medical	BioEng
Genetics	1	0.26	0.24	0.26
Ecology	0.26	1	0.27	0.27
Medical	0.24	0.27	1	0.51
BioEng	0.26	0.27	0.51	1

Figure 2: **Diagram of automatic representation generation.** **(A)** Illustration of the workflow. **(B)** Convergence curve of automatic function abstraction. **(C)** Convergence curve of automatic model abstraction. **(D-F)** Confusion matrices on operation distribution (D), product distribution (E), and device distribution (F), between DSLs across domains. Correlation scores are low except the ones along the diagonals, indicating the significant inter-domain distinctions between the resulting DSLs.

unit of product flow through the protocol. We verify that the product is generated by the specified operation and that it possesses the necessary properties ω_t^{prop} for consumption by the next operation.

The interaction between these two threads forms a feedback loop where the verification of operations and products mutually inform and constrain each other. This reciprocative method allows us to iteratively refine the protocol, ensuring that each step is both operationally feasible and chemically coherent. LLMs are employed to implement the functions CHECKOPCONDITIONS and CHECKPROPERTIES, extracting and verifying operation conditions and product properties from natural language protocol descriptions through instruction-following in-context learning (Wei et al., 2021; Brown et al., 2020). For the prompts employed, readers are referred to Appx. D.6.

3 AUTOMATIC REPRESENTATION GENERATION

In this section, we describe the proposed data-driven algorithm to automatically generate the hierarchically encapsulated representation for protocol design (see Fig. 2A). We first define the problem of generating the desired representation by means of DSL design (Sec. 3.1). We then introduce methods for generating operation-centric (Sec. 3.2) and product-flow-centric (Sec. 3.3) DSL views.

3.1 THE REPRESENTATION GENERATION PROBLEM

We denote the problem of generating the representation for protocol design within a given domain as RG $= (\{\langle\varphi\rangle, \langle\omega\rangle\} \mid \mathcal{P}, \mathcal{C})$. The representation is a DSL with language features accommodating both the operation-centric program view $\langle\varphi\rangle$ and the product-flow-centric program view $\langle\omega\rangle$. The domain-specific corpus $\mathcal{C} = \{\Phi_1, \Phi_2, \ldots, \Phi_{|\mathcal{C}|}\}$ consists of **existing** protocols published in top-quality journals within the corresponding experimental domain. The source and profiles of $\mathcal{C}$ of each domain is detailed in Appx. E.1. We can obtain instance action with attributes based on $\mathcal{C}$ in a straightforward way through NL information extraction (see Appx. D.3 for implementation details). The prior knowledge of operations and products, $p(\varphi)$ and $p(\omega)$, including the basic syntax of the key-value structures and the elementary taxonomies, is derived according to the general commonsense of experimental sciences, as aforementioned in Sec. 2. Specifically, the problem essentially aims to fit the joint distribution models $p(\varphi, \phi, \varphi^{\text{prec}}, \varphi^{\text{post}}, \varphi^{\text{exec}})$ and $p(\omega, \omega^{\text{pred}}, \omega^{\text{succ}}, \omega^{\text{prop}})$ with domain-specific corpus $\mathcal{C}$ given prior knowledge $p(\varphi)$ and $p(\omega)$.

3.2 AUTOMATIC FUNCTION ABSTRACTION

The key challenge of encapsulating the operation-centric view is to aggregate all possible execution contexts for an operation, and then generalize the contexts to the interface. If we keep each of the use case as one single instance of the interface, which can be in thousands regarding one operation, the generalization is meaningless. Since there is no prior knowledge about the interface in advance, we develop the algorithm following the idea of non-parametric modeling, *i.e.*, Dirichlet Process Mixture Model (DPMM), resulting in flexible identification of interface instances.

Hierarchical non-parametric modeling As we must handle information coming in different granularities, from interface structures to values of parameters, we choose to model the operations in a hierarchical fashion. Compared with the flatten spectral clustering approach developed by Shi et al. (2024a), which compresses all information of an operation into a embedding vector, our modeling is competent for considering information at different levels comprehensively. We carefully adopt the prerequisite that the interface is generated subject to the operation, preconditions, postconditions, and execution configurations are generated subject to the interface, and the value of configuration parameters are generated subject to their corresponding keys. Thus, we have the model:

$$\begin{aligned}&p(\varphi, \phi, \varphi^{\text{prec}}, \varphi^{\text{post}}, \varphi^{\text{exec}}, \varphi^{\text{exec-v}})\\&= p(\varphi^{\text{exec-v}} \mid \varphi, \phi, \varphi^{\text{exec}}) p(\varphi^{\text{exec}} \mid \varphi, \phi) p(\varphi^{\text{prec}} \mid \varphi, \phi) p(\varphi^{\text{post}} \mid \varphi, \phi) p(\phi \mid \varphi) p(\varphi),\end{aligned} \tag{1}$$

where $\varphi^{\text{exec-v}}$ denotes the values of configuration parameters. Within each iteration of the DPMM process, we sample the variables level-by-level. Since the structures of preconditions, postconditions, and the selection of devices and configuration parameters are discrete, we sample them directly from the Dirichlet Process. As permissible values of parameters can be discrete, *e.g.*, an array of specific values, common in acidity preparation; continuous, *e.g.*, an interval with minimum and maximum values, common in temperature setting; or mixed, *e.g.*, an array of specific values with random perturbations around the mean, common in timing control, we conduct the sampling by integrating Gaussian Process with Dirichlet Process $\varphi^{\text{exec-v}} \mid \varphi, \phi, \varphi^{\text{exec}} \sim DP(\alpha, H(\varphi^{\text{exec}}), \phi, \varphi) \times GP(m, K)$, where α, H, m, and K are corresponding hyperparameters.

Unification of the interface While clustering similar interface instances encapsulates operations, there may remain redundant interfaces due to minor discrepancies. These discrepancies often arise from differences in parameter values or naming conventions that do not fundamentally alter the operation's functionality. To alleviate such redundancies, we implement a unification process for the interfaces. Specifically, interface instances associated with the same operation are considered equivalent if they have the same number of slots and emits and share the same keys in their execution configuration parameters. By abstracting away differences in parameter values and names, we unify these interfaces into a single, generalized interface, akin to the algorithm proposed by Martelli & Montanari (1982). Unification enhances the generality of the operation-centric view by consolidating functionally-identical interfaces, maintaining a concise and representative set of operations.

Results Function abstraction converges on the domains respectively, as shown by the likelihood curve yielded by non-parametric model in Fig. 2B. In the DSL of Genetics, there are 304 operations in total, with an average of 7.9 interface instances per operation; for Medical, these two quantities are 269 and 6.9; for Bioengineering, they are 196 and 7.8; and for Ecology, they are 100 and 3.5. We find that a majority of operations with high occurrence frequency are unique to one domain, such as `Pipette` to Medical and `Lyse` to Genetics (see Fig. 2D). There are also common operations across domains, such as `Concentrate` and `Culture`. Take `Concentrate` for an example, its interface captures the instances with different devices according to input phases, *e.g.*, use `Bench-top_centrifuge` for `Liquid` while `Isotope_separation_centrifuge` for `Gas`, and also instances with different emits, *e.g.*, selecting `Supernatant` or `Suspension` as the product to keep.

3.3 AUTOMATIC MODEL ABSTRACTION

The key challenge of encapsulating the product-flow-centric view is to select proper descriptive properties of a flow unit component. There exists false positive cases, where properties are attributed to components with the same semantic identifier but in different phases, *e.g.*, we consider ethanol with the property volume when it comes in liquid and with the property pressure when it comes in gas. There also exists false negative cases, where exact same components are regarded as different ones due to different reference names, *e.g.*, Acetylsalicylic Acid, ASA, and Aspirin refer to the same thing. To alleviate false positive and false negative results, we discard the design choice of the interface in the operation-centric view, which tends to cover the possibly richest context, and thereby have the non-parametric model:

$$\begin{aligned}&p(\omega, \omega^{\text{pred}}, \omega^{\text{succ}}, \omega^{\text{prop}}, \omega^{\text{prop-v}})\\&= p(\omega^{\text{prop-v}} \mid \omega^{\text{prop}}, \omega) p(\omega^{\text{prop}} \mid \omega) p(\omega^{\text{pred}} \mid \omega) p(\omega^{\text{succ}} \mid \omega) p(\omega),\end{aligned} \tag{2}$$

where $\omega^{\text{prop-v}}$ denotes the values of property parameters.

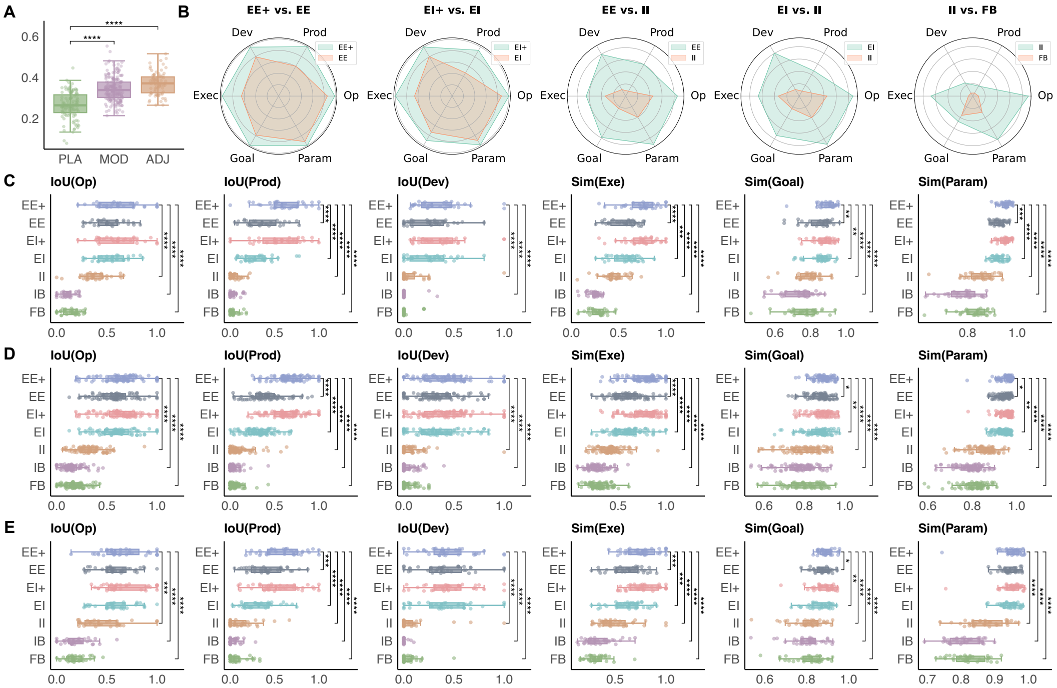


Figure 3: **Results of protocol design. (A)** Profile of text-level similarity between testing sets of the three tasks. **(B)** Pairwise comparison between the capabilities of different machine designers across the six dimensions. **(C-E)** Performances of the seven machine designers on the planning (C), modification (D), and adjustment (E) tasks across the six dimensions (index by column).

Results Model abstraction converges on the domains respectively, as shown by the likelihood curve in Fig. 2C. In the DSL of Genetics, there are $17,190$ model states, *i.e.*, product flow unit as product status, in total; the quantity is $12,472$ for Medical; $11,418$ for Bioengineering; and $2,205$ for Ecology. We find that most components of product flow units with high occurrence frequency are unique to one domain, such as `RNA` to Genetics and `HCC` to Medical (see Fig. 2E/F). Take `Ethanol` for example, the model captures its possible concentrations in liquid rather than in gas.

4 Experiments and Discussion

In this section, we report and discuss the results of our experiments. We start from describing our realistic novel protocol design tasks (Sec. 4.1), along with the metrics to measure the consistency between the designed protocol and the groundtruth protocol (Sec. 4.2). Afterwards, we introduce the alternative representations and machine designers used for comparison (Sec. 4.3). Finally, we report and analyze the experimental results both quantitatively and qualitatively (Sec. 4.4).

4.1 Protocol Design Tasks

Generating unverified experimental objectives and their corresponding protocols specially for our protocol design tasks is impractical because those experiments which have not been peer-reviewed and published can be problematic regarding the contents themselves. To maintain both reality and scale of the testing set, for each domain we **filter out** a small subset of protocols which significantly differ from the remaining major part of the protocol set and **exclude** this subset from the corpora for automatic representation generation (Appx. E.1). This selected subset form the groundtruth of the testing set.

Table 1: **Statistics of the testing set.** Each cell presents the total number of protocols m and experimental steps n in the form m (n).

	Genetics	Medical	Bioengineering	Ecology
Planning	10 (130)	7 (96)	12 (157)	2 (25)
Modification	37 (442)	15 (225)	16 (210)	6 (59)
Adjustment	23 (219)	5 (87)	2 (26)	5 (81)

We exploit quantitative indicators to assist testing set selection, which follows the convention of measuring a protocol's *novelty* in experimental sciences (Schwab & Held, 2020). We comprehensively consider three indicators: (i) similarity between the text embedding of the NL-based description of purpose of protocols, employing the evaluation model in O'Donoghue et al. (2023); (ii) Intersection over Union (IoU) between the instance actions of protocols; (iii) similarity between the execution sequence of protocols, implemented through the Sequence Alignment (SA) algorithm (Smith et al., 1981). To note, indicators (ii) and (iii) are calculated upon the protocols pre-processed by the workflow described in Appx. D.3. Indicator (i) captures the high-level idea of protocol design, indicator (ii) is correlated to the implementation of the protocol design, while indicator (iii) captures the low-level information of protocol execution.

In response to the three purposes of protocol design introduced in Sec. 1, we specify the planning, modification, and adjustment tasks of protocol design. Candidate planning tasks, which are the confirmation of unverified experimental goals, come with relatively low scores (within the 20% lowest) on indicators (i) and (ii). Candidate modification tasks come with fair scores (around the 40% lowest) on indicators (i) and (ii) and relative low score on indicator (iii). Candidate adjustment tasks come with relatively high scores (within the 40% highest) on all of the three indicators.

We obtain the final testing set through a human-machine collaborative workflow. We first detect the outliers of the original protocol corpus of each domain under the metrics above, thereby forming a candidate set. Afterwards, experts of the corresponding domain (holding at least a Master's degree majoring in that domain) manually check the applicability of protocols in the candidate set with cross-validation, discarding the misclassified ones, requesting for more candidate protocols, and refining the groundtruth file when necessary. The testing set includes 140 new protocols and 1757 steps in total, across the domains of Genetics, Medical, Bioengineering, and Ecology, with 23% for planning, 52% for modification, and 25% for adjustment (see Tab. 1 and Fig. 3A for details).

4.2 Inter-protocol consistency metrics

Evaluating the consistency between a designed protocol and the groundtruth is not like comparing between two plain strings (O'Donoghue et al., 2023). Based on the corresponding commonground in experimental sciences (Bartley et al., 2023), we design six-dimensional metrics to comprehensively cover all of the major factors without biased weighting and composition. The six dimensions include: (i) **IoU on operations**, $\text{IoU(Op)} = \text{IoU}(\{\varphi_{1\ldots|\Phi|}\}, \{\varphi'_{1\ldots|\Phi'|}\})$, IoU between instance actions of the designed protocol Φ and the groundtruth Φ'; (ii) **IoU on reagents and intermediate products**, $\text{IoU(Prod)} = \text{IoU}(\{\omega_{0\ldots|\Phi|}\}, \{\omega'_{0\ldots|\Phi'|}\})$; (iii) **IoU on devices**, $\text{IoU(Dev)} = \text{IoU}(\{\varphi(\text{Dev})_{1\ldots|\Phi|}\}, \{\varphi(\text{Dev})'_{1\ldots|\Phi'|}\})$, where $\varphi(\text{Dev})_t$ denotes the exact device for conducting the instance action φ_t; (iv) **Similarity between the execution sequences**, $\text{Sim(Exec)} = \text{SeqAlign}(\langle\varphi_{0\ldots|\Phi|}\rangle, \langle\varphi'_{0\ldots|\Phi'|}\rangle)$, where $\text{SeqAlign}(\cdot,\cdot)$ denotes the ordered sequence similarity score calculation by the SA algorithm; (v) **Similarity between experimental objectives**, $\text{Sim(Goal)} = \text{Cos}(S(\rho), S(\rho'))$, where $S(\cdot)$ represents the serialization operation on structural representations of protocols; (vi) **Similarity between complete protocols at parameter-wise level**, $\text{Sim(Param)} = \text{Cos}(S(\Phi), S(\Phi'))$. These six dimensions capture protocol information from low to high granularities, and also measure the consistency of both ingredient knowledge and procedural knowledge, offering a relatively objective evaluation standard.

4.3 Machine designers

We implement an array of designers by combining different representations with different LLM-based automatic designers under tractable computing load (see Appx. D.7). We investigate four types of representations, including the original NL-based protocol representation (`Flatten`) and the three levels of encapsulation described in Sec. 2, *i.e.*, instance actions with attributes (`Instance`), operation-centric view only (`Encapsulated`), and the dual representation with operation- and product-flow-centric views (`Encapsulated+`). We consider three types of LLM-based protocol designers: (i) `Baseline`, a pure LLM-based approach with Retrieval-Augmented Generation (RAG) on the corresponding corpora (Appx. D.4); (ii) `Internal`, which takes the specific representation as part of the prompt of an LLM, requesting it to output the protocol under the constraint of the given representation (Appx. D.5); (iii) `External`, where the representation serves as an external constraint layer for the output of an LLM, verifying and refining the designed protocols (Appx. D.6). Notably, the external verifier is part of the resulting DSL as our proposed representation for protocol design.

The combination of representation and designer does not span a Cartesian space due to the intrinsic limitations of Flatten and Baseline. Therefore, we implement seven machine designers, including: (i) Flatten-Baseline(FB), LLM with RAG on original protocol corpora; (ii) Instance-Baseline(IB), LLM retrieval on the protocol corpora translated into instance actions; (iii) Instance-Internal(II), prompting LLM with the Instruction Set Assembly (ISA) of instance actions, following the implementation of the currently state-of-the-art method O'Donoghue et al. (2023); (iv) Encapsulated-Internal(EI), prompting LLM with the DSL with operation-centric view; (v) Encapsulated-External(EE), LLM equipping with the external verifier provided by the DSL with operation-centric view; (vi) Encapsulated-Internal+(EI+), prompting LLM with the DSL with the dual representation; and (vii) Encapsulated-External+(EE+), LLM equipping with the external verifier provided by the DSL with the dual representation.

4.4 Protocol design results

The complete quantitative results across the four domains, the three tasks, and the six dimensions of evaluation metrics are presented at Appx. B. Through paired samples t-test, we find that EE+ and EI+ significantly outperform other alternative approaches (EE+ outperforms EE: $t(278) = 8.007, \mu_d < 0, p < .0001$; EI+ outperforms EI: $t(278) = 8.397, \mu_d < 0, p < .0001$; EE+ outperforms II: $t(278) = 24.493, \mu_d < 0, p < .0001$; EI+ outperforms II: $t(278) = 23.855, \mu_d < 0, p < .0001$; see Fig. 3C-E). These comparisons demonstrate the suitability of our desired representation for protocol design. Similarly, we find that approaches equipping with a relatively higher-level representation significantly outperforms their counterparts with a relatively lower-level representation (EE outperforms II: $t(278) = 16.315, \mu_d < 0$, $p < .0001$; EI outperforms II: $t(278) = 15.259, \mu_d < 0$, $p < .0001$; II outperforms FB: $t(278) = 8.340, \mu_d < 0, p < .0001$; see Fig. 3B).

4.5 Discussion

This work proposes a hierarchically encapsulated representation for the conceptual knowledge in experimental sciences, including instance actions with attributes, sequential representation of operations with function abstraction, and continuous representation of product-flows with model abstraction, to fully elicit LLMs' capability on protocol design as an auxiliary module. The following discussions on results reveal the design rationality, scalability, and generality of the representation.

Contributions of the building blocks The encapsulated representation approaches with dual views outperform their counterparts without dual views by enhancing both intra-step and inter-step details. At the intra-step level, EI and EE offer richer semantic information than IB and II, leveraging protocol-centric view to capture detailed configuration each operation. This feature accounts for their satisfactory performance on IoU(Op). At the inter-step level, EI+ and EE+ treat each step as a FlowUnit, incorporating both preceding and succeeding step contexts, leading to notable improvements in Sim(Exec) and IoU(Prod). This creates a *double assurance* mechanism (Shi et al., 2024c): the first assurance comes from internal input/output checks within each instruction, and the second from the input/output characteristics inferred from neighboring instructions. Namely, we estimate the output of the preceding operation and check its alignment with the current step's input. This design enhances step linkage, verification, and overall coherence, ensuring higher consistency and robustness in complex protocol workflows. Please refer to Appx. I.1 for the case study.

Handling different task complexities The overall performance aligns with the trend in complexity across the three tasks (Fig. 3A); however, the dual-view encapsulated representations, EI+ and EE+, demonstrate superior performance compared to their counterparts. In planning, these methods consider all necessary components, enabling creative yet structured protocol generation. For modification tasks, they provide feedback on parameter changes, detecting inconsistencies that their counterparts might fail to capture. In adjustment tasks, EE+'s external verifier maintains protocol integrity by identifying component relationships. Please refer to Appx. I.2 for the case study.

Generality across domains Our DSL-based approaches offer a unified, modular representation with generalizability across scientific domains (see domain-indexed results at Appx. B.2). The dual-view approach abstracts experimental processes into operations and flow units, capturing essential details while remaining applicable across fields. By representing dependencies between steps and tracking product flow, the replication of experiments could be enhanced. The framework captures cross-domain commonalities while allowing domain-specific content like specialized operations and reagents. This unified representation standardizes protocols and enables researchers to adopt experimental protocols from multiple fields, fostering interdisciplinary collaboration and innovation. Please refer to Appx. I.3 for the case study. Limitations on generality are discussed at Appx. G.

ACKNOWLEDGEMENTS

This work was partially supported by the National Natural Science Foundation of China under Grants 52475001. Q. Xu is a visiting student at Peking University from University of Science and Technology of China. The authors would like to thank Haofei Hou for his earlier works regarding domain-specific representations, and also Jiawen Liu for her assistance in figure drawings.

REFERENCES

Harold Abelson and Gerald Jay Sussman. *Structure and interpretation of computer programs*. The MIT Press, 1996.

Microsoft Research AI4Science and Microsoft Azure Quantum. The impact of large language models on scientific discovery: a preliminary study using gpt-4. *arXiv preprint arXiv:2311.07361*, 2023.

Monya Baker. 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604), 2016.

Monya Baker. Five keys to writing a reproducible lab protocol. *Nature*, 597(7875):293–294, 2021.

Bryan Bartley, Jacob Beal, Miles Rogers, Daniel Bryce, Robert P Goldman, Benjamin Keller, Peter Lee, Vanessa Biggers, Joshua Nowak, and Mark Weston. Building an open representation for biological protocols. *ACM Journal on Emerging Technologies in Computing Systems*, 19(3):1–21, 2023.

Anne-Catherine Bédard, Andrea Adamo, Kosi C Aroh, M Grace Russell, Aaron A Bedermann, Jeremy Torosian, Brian Yue, Klavs F Jensen, and Timothy F Jamison. Reconfigurable system for automated optimization of diverse chemical reactions. *Science*, 361(6408):1220–1225, 2018.

Margaret Boden. Artificial intelligence and natural man. *Synthese*, 43(3), 1980.

Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, 2020.

Benjamin Burger, Phillip M Maffettone, Vladimir V Gusev, Catherine M Aitchison, Yang Bai, Xiaoyan Wang, Xiaobo Li, Ben M Alston, Buyi Li, Rob Clowes, et al. A mobile robotic chemist. *Nature*, 583(7815):237–241, 2020.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. *arXiv preprint arXiv:2202.07646*, 2022.

Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

Thomas L Griffiths. Understanding human intelligence through human limitations. *Trends in Cognitive Sciences*, 24(11):873–883, 2020.

Julie Hirtz, Robert B Stone, Daniel A McAdams, Simon Szykman, and Kristin L Wood. A functional basis for engineering design: reconciling and evolving previous efforts. *Research in Engineering Design*, 13:65–82, 2002.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Longman Publishing Co., Inc., 1996.

Genki N Kanda, Taku Tsuzuki, Motoki Terada, Noriko Sakai, Naohiro Motozawa, Tomohiro Masuda, Mitsuhiro Nishida, Chihaya T Watanabe, Tatsuki Higashi, Shuhei A Horiguchi, et al. Robotic search for optimal cell culture in regenerative medicine. *Elife*, 11:e77007, 2022.

Artem I Leonov, Alexander JS Hammer, Slawomir Lach, S Hessam M Mehr, Dario Caramelli, Davide Angelone, Aamir Khan, Steven O'Sullivan, Matthew Craven, Liam Wilbraham, et al. An integrated self-optimizing programmable chemical synthesis and reaction engine. *Nature Communications*, 15(1):1240, 2024.

Barbara Liskov. Keynote address-data abstraction and hierarchy. In *Addendum to the proceedings on Object-oriented programming systems, languages and applications (Addendum)*, 1987.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, pp. 1–11, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, 2023.

J Sebastián Manzano, Wenduan Hou, Sergey S Zalesskiy, Przemyslaw Frei, Hsin Wang, Philip J Kitson, and Leroy Cronin. An autonomous portable platform for universal chemical synthesis. *Nature Chemistry*, 14(11):1311–1318, 2022.

Alberto Martelli and Ugo Montanari. An efficient unification algorithm. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(2):258–282, 1982.

John McCarthy. *Programs with common sense*. London, 1959.

Marcia McNutt. Reproducibility. *Science*, 343(6168):229–229, 2014.

S Hessam M Mehr, Matthew Craven, Artem I Leonov, Graham Keenan, and Leroy Cronin. A universal system for digitization and automatic execution of the chemical synthesis literature. *Science*, 370(6512):101–108, 2020.

Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, 37(4):316–344, 2005.

Stephen Monsell. Task switching. *Trends in Cognitive Sciences*, 7(3):134–140, 2003.

Allen Newell. The knowledge level. *Artificial Intelligence*, 18(1):87–127, 1982.

Odhran O'Donoghue, Aleksandar Shtedritski, John Ginger, Ralph Abboud, Ali Ghareeb, and Samuel Rodriques. Bioplanner: Automatic evaluation of llms on protocol planning in biology. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

Shubham Pateria, Budhitama Subagdja, Ah-hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

Simon Rohrbach, Mindaugas Šiaučiulis, Greig Chisholm, Petrisor-Alin Pirvan, Michael Saleeb, S Hessam M Mehr, Ekaterina Trushina, Artem I Leonov, Graham Keenan, Aamir Khan, et al. Digitization and validation of a chemical synthesis literature database in the chempu. *Science*, 377(6602):172–180, 2022.

Gilbert Ryle and Julia Tanney. *The concept of mind*. Routledge, 1949.

Simon Schwab and Leonhard Held. Different worlds confirmatory versus exploratory research. *Significance*, 17(2):8–9, 2020.

Yu-Zhe Shi, Shiqian Li, Xinyi Niu, Qiao Xu, Jiawen Liu, Yifan Xu, Shiyu Gu, Bingru He, Xinyang Li, Xinyu Zhao, et al. PersLEARN: Research Training through the Lens of Perspective Cultivation. In *Annual Meeting of the Association for Computational Linguistics*, 2023a.

Yu-Zhe Shi, Manjie Xu, John E Hopcroft, Kun He, Joshua B Tenenbaum, Song-Chun Zhu, Ying Nian Wu, Wenjuan Han, and Yixin Zhu. On the complexity of Bayesian generalization. In *International Conference on Machine Learning*, 2023b.

Yu-Zhe Shi, Haofei Hou, Zhangqian Bi, Fanxu Meng, Xiang Wei, Lecheng Ruan, and Qining Wang. AutoDSL: Automated domain-specific language design for structural representation of procedures with constraints. In *Annual Meeting of the Association for Computational Linguistics*, 2024a.

Yu-Zhe Shi, Haotian Li, Lecheng Ruan, and Huamin Qu. Constraint representation towards precise data-driven storytelling. In *IEEE Visualization and Visual Analytics Gen4DS Workshop*, 2024b.

Yu-Zhe Shi, Fanxu Meng, Haofei Hou, Zhangqian Bi, Qiao Xu, Lecheng Ruan, and Qining Wang. Expert-level protocol translation for self-driving labs. In *Advances in Neural Information Processing Systems*, 2024c.

Yu-Zhe Shi, Qiao Xu, Fanxu Meng, Lecheng Ruan, and Qining Wang. Abstract Hardware Grounding towards the Automated Design of Automation Systems. In *International Conference on Intelligent Robotics and Applications*, 2024d.

Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

Sebastian Steiner, Jakob Wolf, Stefan Glatzel, Anna Andreou, Jarosław M Granda, Graham Keenan, Trevor Hinkley, Gerardo Aragon-Camarasa, Philip J Kitson, Davide Angelone, et al. Organic synthesis in a modular robotic system driven by a chemical programming language. *Science*, 363 (6423):eaav2211, 2019.

Nathan J Szymanski, Bernardus Rendy, Yuxing Fei, Rishi E Kumar, Tanjin He, David Milsted, Matthew J McDermott, Max Gallant, Ekin Dogus Cubuk, Amil Merchant, et al. An autonomous laboratory for the accelerated synthesis of novel materials. *Nature*, 624(7990):86–91, 2023.

Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A Saurous, and Yoon Kim. Grammar prompting for domain-specific language generation with large language models. In *Advances in Neural Information Processing Systems*, 2023a.

Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, et al. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972):47–60, 2023b.

Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021.

Tianwen Wei, Liang Zhao, Lichang Zhang, Bo Zhu, Lijie Wang, Haihua Yang, Biye Li, Cheng Cheng, Weiwei Lü, Rui Hu, et al. Skywork: A more open bilingual foundation model. *arXiv preprint arXiv:2310.19341*, 2023.

Ziyang Xiao, Dongxiang Zhang, Yangjun Wu, Lilin Xu, Yuan Jessica Wang, Xiongwei Han, Xiaojin Fu, Tao Zhong, Jia Zeng, Mingli Song, et al. Chain-of-experts: When llms meet complex operations research problems. In *International Conference on Learning Representations*, 2023.

A Additional remarks

A.1 Rationale of the overall design choice

It seems that we can formulate the protocol design problem in the fashion of Markov Decision Process (MDP) and solve it by heuristic-based planning methods or Hierarchical Reinforcement Learning (HRL) approaches. However, although the formulation itself is feasible, solving the problem may not be practical. Consider solving the problem through an HRL approach designed for heterogeneous action space with parameters (as the protocol is required to decide both the key properties of an operation and the corresponding values). This hierarchical agent may be trained to converge on a fine-grained environment with a clearly designed reward function, or on a large dataset with trajectories for offline learning. Unfortunately, we have access to neither an interactive environment simulating the experiments nor sufficient data to support offline training (Pateria et al., 2021).

Treating the experimental procedures as a white box and creating digital twins for experiments can be an elegant solution and thereby facilitate various applications other than protocol design. This effort requires elaborated design of simulation granularity, exhaustive collection of primitive principles of the system, efficient implementation of rule production, and define precise metrics for evaluating the distance between current and objective states (serving as a reward function), which can be labor-intensive and is far out of the scope of this work. On the other hand, viewing those published protocols as trajectories for offline training, the scale of the offline dataset and the density of the reward function are much too insufficient to support training to convergence. Augmenting the data, synthesizing realistic trajectories, or enhancing the accessibility of protocols, are out of the scope of this work. Given the current obstacles, we choose not to formulate the problem in an MDP fashion. Though an MDP-style formulation can be more precise and elegant, it may misguide the readers to some extent. Instead, we decide to leverage the rich domain-specific knowledge provided by knowledge-based agents such as LLMs, where knowledge may complement the lack of data and dense reward function. This design choice is also in line with the initial attempts on automatic experiment design (Boiko et al., 2023; M. Bran et al., 2024).

In summary, our design choice of formulation is a compromise based on currently limited resources and restricted scope. Nonetheless, the exploration of more precise and elegant formulations represents a promising avenue for future research.

A.2 Intuition behind the interface

Interface is a concept of functional abstraction (Abelson & Sussman, 1996). Interface disentangles the abstract functionality on the semantics level and its corresponding implementation details on the execution level. This approach encapsulates the implementation of an operation into a *black-box*, so the users of the operation would only need to consider its input and output. Therefore, with such encapsulated representation for protocol design, we only need to care about the consistency between the output of the predecessor operation and the input of the successor operation, without caring about their implementation details.

This is the idea behind operationalization. Operationalization makes the interface an abstract function over all relative instance actions. The interface is abstracted from the execution contexts of all instance actions with the same reference name, *i.e.*, the same purpose, and can be instantiated to an instance action given a specific execution context. A specific context can be the predecessor operation, the successor operation, the precondition, or the postcondition of the considered operation. An instance action configures a specific implementation for a specific execution context. For the operation *"Homogenization"*, the implementation of one instance action can be *"using an ultrasonic homogenizer"* if the precondition, namely, the execution context, has intermediate product *"cell suspension"* available; the implementation of another instance action can be *"using a bead mill"* if the precondition contains tissue. This example demonstrates the relationship between interface and instance actions of an operation: the interface is abstracted from the set of instance actions and can be instantiated to instance actions.

Here we also give a more intuitive example to enhance the reader's comprehension. Consider the culinary scenario with the actions *"frying the egg"*, *"frying the fish"*, and *"frying the steak"*.These are different instance actions coming with the same purpose *"to fry something"*. Therefore, we can abstract the interface from these instance actions to operationalize the operation *"fry"*. The input of *"fry"* should be something raw and its output should be something fried. Given different preconditions with available eggs or pieces of steak, the abstract semantic operation *"fry"* can be grounded to instance actions *"frying the egg"* or *"frying the steak"* respectively, through the instantiation of the interface. In summary, an interface serves as the bridge between the semantics level and the execution level.

A.3 Values of manual protocol certification

Certification is always one of the central focuses in the engineering practices of automation. In our practice, we only automate the process of protocol design, which is the primary objective of this work, and keep the manual certification part. On one hand, relieving experimental scientists from the labour-intensive protocol design tasks, thereby allowing them more time for high-level thinking, is a sufficiently significant improvement so far. On the other hand, engineering practices such as lab automation and manufacturing are in high demand for preciseness. This leads to the requirement of manual certification. Domain experts handle subtle cases through their tacit domain-specific knowledge and are responsible for their decisions (Wang et al., 2023b). According to these considerations and the standard operating processes of experimental sciences, we choose to certify the designed protocols by domain experts.

Our current choice is a compromise on the limitation of techniques and the demand for preciseness. In future work, we can conduct investigations on how to build digital twins of self-driving laboratories. Such digital twins support prediction, explanation, and counterfactual analysis of unseen behaviors of the experiments, which may facilitate machine-based protocol certification. Grounding these blue-sky thoughts necessitates addressing the challenging problems regarding the decision of simulation granularity, the implementation of data-efficient simulation model construction, and the injection of tacit domain-specific knowledge. In summary, the exploration of generated-protocol-certification by machines represents a promising avenue for future research.

A.4 Limitations of automatic protocol certification

LLMs can be much too uncontrollable for engineering practices such as lab automation, which may lead to unpredictable dangerous situations (Wang et al., 2023b). There comes a dilemma—we try to exploit the capability of reasoning over knowledge of LLMs, while we try to alleviate the drawbacks brought up by the uncontrollable nature of LLMs. Our proposed representation is dedicated to resolving the dilemma. The representations not only elicit LLMs' potential on protocol design through structural knowledge representation, but also serve as a guardrail for LLMs. Since the generated protocols are represented as corresponding DSL programs, the permissible output space is much more confined compared with that of pure LLMs, serving as constraints upon the LLM-generated protocols. Thanks to the verification mechanisms provided by DSLs, the correctness of the generated protocols can be checked to some extent. Therefore, by equipping LLMs with an auxiliary constraint layer, we may approach a balance between knowledge utilization and preciseness.

However, the current verification on the level of DSL programs is far from sufficient for serving as a certification. Certification is a serious process, where any possibilities of reporting false positive cases are required to be eliminated. Some cases can be highly long-tailed distributed, which may not be detected by data-driven and knowledge-driven machine certifiers. In this context, human domain experts are responsible for coming up with these potential risks through their experiences and tacit knowledge. Therefore, we are not likely to move human experts out of the loop, except that we can efficiently build up appropriate digital twins for self-driving laboratories. In current practices, the automation of protocol design puts human experts into a larger loop without focusing on the low-level details of experiments. As a result, they are allowed more time for high-level thoughts on things like values, which are not likely to be alternated by machines. In summary, it is neither practical nor necessary to totally move human experts out of the loop of automatic scientific discovery. The investigation of human-machine coordination in protocol certification represents a promising avenue for future research.

A.5 Rationale for the reagent consumption model

We treat the instantiation and the consumption of reagents a *one-time deal* without considering the exact volume of consumption and the corresponding remainder. The rationale for such design choice comes from both the current Standard Operating Process (SOP) of experimental sciences and the properties of self-driving laboratories (Bartley et al., 2023).

In the current SOP for manually conducted experiments, experimenters are required to use prefabricated sets of reagents. Similarly, experimenters use specific containers with predefined capacities to transfer intermediate products. Therefore, one pack of reagents or one container of intermediate products is only used once for an operation, without considering the remainder. This results in a more succinct representation where reagents are regarded as discrete elements rather than continuous volumes.

For self-driving laboratories, this is deliberately designed for efficient variable management following the corresponding principles in computer system design (Abelson & Sussman, 1996). In computer systems, not removing used variables would cause out-of-memory errors, let alone in physical automation systems, where the physical memory slots are much harder than the virtual memory slots in computer systems to manage. Hence, we exploit this variable management mechanism to enhance the execution efficiency of self-driving laboratories.

A.6 Relation to LLM reasoning

We would like to clarify that our objective is not to alternate Chain-of-Thought (CoT) reasoning. According to recent studies on the properties of CoT, LLMs with CoT may generate coherent but unprofessional text in expertise-intensive application scenarios (Xiao et al., 2023). Therefore, our proposed representation serves as an auxiliary guardrail module for LLMs with reasoning techniques such as CoT, enhancing LLMs' reasoning capability from two aspects: (i) the representation constrain the scope of reasoning into a close set of entities, such as available operations, reagents, and devices commonly used in the domain; and (ii) the representation provides fine-grained injection of domain-specific knowledge for LLMs, resulting in not only coherent but also expertise-compatible generated content.

A.7 Applicability to domains beyond scientific experiment

In theory, our framework can be applied to any field that requires adherence to specific protocols and has a need for automated execution. Let us consider an automated kitchen controlled by a computer as an example.

Assuming the automated kitchen's computer is already programmed to prepare "braised pork ribs" and "steamed sea bass":

```
Braised Pork Ribs:

1.Select pork ribs as the main ingredient.
2.Heat a pan over high heat.
3.Add the ribs to the pan and fry for about 5 minutes until they are
    browned.
4.Add seasonings: soy sauce and sugar.
5.Reduce the heat to medium.
6.Simmer the ribs for 30 minutes until tender.
7.Serve hot.

START
SELECT ingredient: ribs
ACTION: fry, temperature: high, time: 5 min
ADD seasoning: soy sauce, sugar
ACTION: simmer, temperature: medium, time: 30 min
END

Steamed Sea Bass:

1.Select a whole sea bass as the main ingredient.
2.Prepare a steamer and heat it to high temperature.
```

```
3.Place the sea bass in the steamer.
4.Steam the fish for about 15 minutes until fully cooked.
5.Add seasonings: ginger slices and chopped scallions.
6.Serve immediately with the garnish.

START
SELECT ingredient: sea bass
ACTION: steam, temperature: high, time: 15 min
ADD seasoning: ginger, scallion
END
```

Next, we can derive the corresponding DSL. For instance:

```
{
  "cooking_methods": {
    "braise": {
      "steps": [
        {"type": "fry", "temperature": "high", "time": "5 min"},
        {"type": "simmer", "temperature": "medium", "time": "30 min"}
      ],
      "seasoning": ["soy sauce", "sugar"]
    },
    "steam": {
      "steps": [
        {"type": "steam", "temperature": "high", "time": "15 min"}
      ],
      "seasoning": ["ginger", "scallion"]
    }
  },
  "ingredients": {
    "ribs": {
      "category": "meat",
      "default_braise_time": "30 min"
    },
    "sea_bass": {
      "category": "fish",
      "default_braise_time": "20 min",
      "default_steam_time": "15 min"
    }
  }
}
```

Now, let us create a new recipe for Braised Sea Bass by combining the braising technique with sea bass as the main ingredient.

```
START
SELECT ingredient: sea bass
ACTION: fry, temperature: high, time: 5 min
ADD seasoning: soy sauce, sugar
ACTION: simmer, temperature: medium, time: 20 min
END
```

B Complete results

B.1 Task-indexed complete results

Table A1: **Complete quantitative results on protocol design, specifically the planning task.** Each cell represents the machine designer's average score (at the top of the cell) on all testing samples across the four domains and the corresponding standard error of mean (at the bottom of the cell). For each dimension, we highlight the results of both **the best** and **the second best** ones.

	IoU(Op)	IoU(Prod)	IoU(Dev)	Sim(Exec)	Sim(Goal)	Sim(Param)
FB	0.143 (0.087)	0.040 (0.055)	0.020 (0.060)	0.282 (0.090)	0.766 (0.096)	0.826 (0.059)
IB	0.109 (0.067)	0.036 (0.053)	0.019 (0.074)	0.242 (0.065)	0.735 (0.090)	0.781 (0.069)
II	0.382 (0.154)	0.050 (0.062)	0.084 (0.191)	0.452 (0.134)	0.788 (0.074)	0.851 (0.062)
EI	0.542 (0.160)	0.305 (0.181)	0.259 (0.211)	0.572 (0.152)	0.849 (0.066)	0.926 (0.026)
EI+	**0.603** (0.208)	**0.555** (0.260)	**0.357** (0.237)	**0.737** (0.172)	**0.875** (0.057)	**0.949** (0.023)
EE	0.524 (0.151)	0.370 (0.198)	0.252 (0.206)	0.558 (0.148)	0.846 (0.078)	0.928 (0.025)
EE+	**0.607** (0.211)	**0.605** (0.235)	**0.355** (0.242)	**0.744** (0.179)	**0.893** (0.056)	**0.951** (0.021)

Table A2: **Complete quantitative results on protocol design, specifically the modification task.** Each cell represents the machine designer's average score (at the top of the cell) on all testing samples across the four domains and the corresponding standard error of mean (at the bottom of the cell). For each dimension, we highlight the results of both **the best** and **the second best** ones.

	IoU(Op)	IoU(Prod)	IoU(Dev)	Sim(Exec)	Sim(Goal)	Sim(Param)
FB	0.181 (0.102)	0.050 (0.071)	0.038 (0.071)	0.304 (0.102)	0.796 (0.090)	0.809 (0.060)
IB	0.150 (0.100)	0.038 (0.065)	0.039 (0.076)	0.281 (0.100)	0.772 (0.089)	0.788 (0.060)
II	0.331 (0.143)	0.101 (0.131)	0.061 (0.135)	0.416 (0.127)	0.802 (0.087)	0.851 (0.059)
EI	0.593 (0.186)	0.318 (0.158)	0.336 (0.235)	0.602 (0.164)	0.866 (0.066)	0.937 (0.030)
EI+	**0.648** (0.210)	**0.626** (0.188)	**0.413** (0.256)	**0.765** (0.170)	**0.883** (0.055)	**0.952** (0.031)
EE	0.588 (0.185)	0.403 (0.192)	0.332 (0.228)	0.601 (0.164)	0.873 (0.053)	0.940 (0.028)
EE+	**0.640** (0.213)	**0.661** (0.179)	**0.410** (0.253)	**0.757** (0.170)	**0.893** (0.043)	**0.953** (0.032)

Table A3: **Complete quantitative results on protocol design, specifically the adjustment task.** Each cell represents the machine designer's average score (at the top of the cell) on all testing samples across the four domains and the corresponding standard error of mean (at the bottom of the cell). For each dimension, we highlight the results of both **the best** and **the second best** ones.

	IoU(Op)	IoU(Prod)	IoU(Dev)	Sim(Exec)	Sim(Goal)	Sim(Param)
FB	0.192 (0.100)	0.077 (0.104)	0.051 (0.094)	0.319 (0.103)	0.811 (0.078)	0.823 (0.051)
IB	0.197 (0.131)	0.039 (0.063)	0.006 (0.021)	0.337 (0.141)	0.802 (0.082)	0.810 (0.049)
II	0.453 (0.208)	0.115 (0.161)	0.091 (0.211)	0.508 (0.184)	0.805 (0.081)	0.873 (0.056)
EI	0.587 (0.190)	0.328 (0.186)	0.400 (0.265)	0.623 (0.165)	0.863 (0.055)	0.944 (0.027)
EI+	**0.668** (0.208)	**0.545** (0.259)	**0.449** (0.247)	**0.775** (0.152)	**0.883** (0.056)	**0.950** (0.040)
EE	0.581 (0.184)	0.404 (0.205)	0.395 (0.261)	0.616 (0.162)	0.875 (0.039)	0.946 (0.026)
EE+	**0.650** (0.220)	**0.589** (0.229)	**0.441** (0.248)	**0.758** (0.160)	**0.893** (0.033)	**0.950** (0.042)

B.2 Domain-indexed complete results

Table A4: **Complete quantitative results on protocol design, specifically the Genetics domain.** Each cell represents the machine designer's average score (at the top of the cell) on all testing samples across the three tasks and the corresponding standard error of mean (at the bottom of the cell). For each dimension, we highlight the results of both **the best** and **the second best** ones.

	IoU(Op)	IoU(Prod)	IoU(Dev)	Sim(Exec)	Sim(Goal)	Sim(Param)
FB	0.179 (0.113)	0.065 (0.082)	0.037 (0.080)	0.301 (0.116)	0.795 (0.091)	0.805 (0.066)
IB	0.157 (0.129)	0.042 (0.060)	0.022 (0.059)	0.297 (0.137)	0.793 (0.070)	0.789 (0.062)
II	0.379 (0.200)	0.120 (0.158)	0.079 (0.160)	0.457 (0.180)	0.807 (0.083)	0.850 (0.072)
EI	0.599 (0.189)	0.332 (0.177)	0.353 (0.243)	0.619 (0.164)	0.862 (0.055)	0.941 (0.026)
EI+	**0.691** (0.198)	**0.606** (0.252)	**0.429** (0.283)	**0.803** (0.151)	**0.882** (0.054)	**0.954** (0.033)
EE	0.592 (0.189)	0.415 (0.206)	0.351 (0.241)	0.615 (0.163)	0.870 (0.052)	0.943 (0.025)
EE+	**0.677** (0.210)	**0.653** (0.228)	**0.425** (0.280)	**0.791** (0.161)	**0.888** (0.045)	**0.955** (0.034)

Table A5: **Complete quantitative results on protocol design, specifically the Medical domain.** Each cell represents the machine designer's average score (at the top of the cell) on all testing samples across the three tasks and the corresponding standard error of mean (at the bottom of the cell). For each dimension, we highlight the results of both **the best** and **the second best** ones.

	IoU(Op)	IoU(Prod)	IoU(Dev)	Sim(Exec)	Sim(Goal)	Sim(Param)
FB	0.174 (0.085)	0.048 (0.070)	0.030 (0.067)	0.312 (0.075)	0.796 (0.087)	0.839 (0.043)
IB	0.139 (0.054)	0.029 (0.038)	0.023 (0.063)	0.264 (0.045)	0.721 (0.123)	0.795 (0.070)
II	0.373 (0.093)	0.081 (0.087)	0.091 (0.205)	0.424 (0.072)	0.776 (0.097)	0.871 (0.041)
EI	0.604 (0.167)	0.322 (0.146)	0.309 (0.253)	0.594 (0.148)	0.861 (0.079)	0.932 (0.031)
EI+	**0.615** (0.196)	**0.574** (0.242)	**0.400** (0.198)	**0.758** (0.149)	0.871 (0.060)	**0.952** (0.021)
EE	0.591 (0.158)	0.373 (0.166)	0.298 (0.234)	0.583 (0.149)	**0.873** (0.054)	0.936 (0.030)
EE+	**0.615** (0.197)	**0.613** (0.210)	**0.390** (0.202)	**0.756** (0.151)	**0.891** (0.040)	**0.955** (0.019)

Table A6: **Complete quantitative results on protocol design, specifically the Ecology domain.** Each cell represents the machine designer's average score (at the top of the cell) on all testing samples across the three tasks and the corresponding standard error of mean (at the bottom of the cell). For each dimension, we highlight the results of both **the best** and **the second best** ones.

	IoU(Op)	IoU(Prod)	IoU(Dev)	Sim(Exec)	Sim(Goal)	Sim(Param)
FB	0.155 (0.085)	0.030 (0.035)	0.021 (0.048)	0.297 (0.088)	0.781 (0.096)	0.807 (0.056)
IB	0.162 (0.118)	0.006 (0.015)	0.030 (0.058)	0.275 (0.105)	0.763 (0.090)	0.788 (0.063)
II	0.386 (0.176)	0.043 (0.062)	0.027 (0.062)	0.448 (0.131)	0.788 (0.065)	0.856 (0.044)
EI	**0.458** (0.171)	0.259 (0.134)	**0.351** (0.195)	0.514 (0.142)	0.879 (0.048)	0.933 (0.028)
EI+	0.411 (0.134)	**0.569** (0.133)	**0.359** (0.175)	**0.586** (0.127)	**0.888** (0.052)	**0.945** (0.023)
EE	**0.458** (0.171)	0.347 (0.151)	**0.351** (0.195)	0.507 (0.138)	0.874 (0.048)	0.934 (0.029)
EE+	0.414 (0.142)	**0.581** (0.141)	0.346 (0.177)	**0.586** (0.131)	**0.910** (0.035)	**0.944** (0.024)

Table A7: **Complete quantitative results on protocol design, specifically the Bioengineering domain.** Each cell represents the machine designer's average score (at the top of the cell) on all testing samples across the three tasks and the corresponding standard error of mean (at the bottom of the cell). For each dimension, we highlight the results of both **the best** and **the second best** ones.

	IoU(Op)	IoU(Prod)	IoU(Dev)	Sim(Exec)	Sim(Goal)	Sim(Param)
FB	0.176 (0.085)	0.048 (0.089)	0.050 (0.081)	0.300 (0.084)	0.790 (0.090)	0.826 (0.042)
IB	0.149 (0.077)	0.050 (0.087)	0.038 (0.091)	0.286 (0.078)	0.767 (0.083)	0.797 (0.046)
II	0.352 (0.151)	0.062 (0.090)	0.066 (0.187)	0.443 (0.125)	0.810 (0.073)	0.860 (0.045)
EI	0.565 (0.164)	0.307 (0.186)	0.310 (0.249)	0.603 (0.169)	0.851 (0.072)	0.930 (0.033)
EI+	**0.657** (0.209)	**0.577** (0.177)	**0.394** (0.241)	**0.743** (0.179)	**0.888** (0.056)	**0.944** (0.041)
EE	0.558 (0.162)	0.392 (0.214)	0.303 (0.246)	0.598 (0.165)	0.855 (0.076)	0.933 (0.030)
EE+	**0.653** (0.206)	**0.614** (0.172)	**0.401** (0.246)	**0.742** (0.176)	**0.900** (0.046)	**0.945** (0.041)

C Ethics statement

C.1 Human expert participants

The testing set selection and groundtruth checking tasks conducted by human experts in this work has been approved by the Institutional Review Board (IRB) of Peking University. We have been committed to upholding the highest ethical standards in conducting this study and ensuring the protection of the rights and welfare of all participants. We paid the domain experts a wage of $22.5/h for their work in this study.

We have obtained informed consent from all human experts, including clear and comprehensive information about the purpose of the study, the procedures involved, the risks and benefits, and the right to withdraw at any time without penalty. Participants were also assured of the confidentiality of their information. Any personal data collected (including name, age, and gender) was handled in accordance with applicable laws and regulations.

C.2 Corpora collection

We carefully ensure that all protocols included in our corpora strictly comply with open access policies under the Creative Commons license. This strategy guarantees adherence to copyright and intellectual property laws, thereby preventing any potential infringement or unauthorized use of protected materials. By exclusively employing resources that are freely accessible and legally distributable, we maintain the highest standards of ethical research conduct, promoting transparency and respect for the intellectual property rights of others. This commitment ensures that our work advances the frontiers of knowledge in a manner that is both legally sound and ethically responsible.

D Implementation details

D.1 Prior model of product flow-centric view

```
<ProductFlow> ::= <Pred> <FlowUnit> <Succ>
```

```

<Pred> ::= <Operation.UniqueName>

<Succ> ::= <Operation.UniqueName>

<FlowUnit> ::= <Component> <ComponentType> <RefName> <Vol> <Container> *<
    Cond>
<Component> ::= <STR>
<ComponentType> ::= Gas | Liquid | Solid | Semi-Solid | Mixture |
    ChemicalCompound | BiologicalMaterial | Reagent | PhysicalObject |
    File/Data | ... [Known component types]
<RefName> ::= <Component> <Index>
<UnitArgType> ::= MAT | PROD
<Vol> ::= <REAL> <MEAS>
<Container> ::= Tube | Flask | Pipette | ... [Known container types]
<Cond> ::= <ArgKey> <ArgValue>
<ArgKey> ::= Temperature | Pressure | Acidity | Lighting | ... [Known
    conditional keys]
<ArgValue> ::= <REAL> <MEAS>
```

D.2 PRIOR MODEL OF OPERATION-CENTRIC VIEW

```
<Operation> ::= <UniqueName> *<Pattern>

<UniqueName> ::= <STR>

<Pattern> ::= <Precond> <Execution> <Postcond> *<Example>

<Precond> ::= <SlotArgNum> *<SlotArg>
<SlotArgNum> ::= <INT>
<SlotArg> ::= <ProductFlow.FlowUnit.ComponentType>

<Postcond> ::= <EmitArgNum> *<EmitArg>
<EmitArgNum> ::= <INT>
<EmitArg> ::= <ProductFlow.FlowUnit.ComponentType>

<Example> ::= <STR>

<Execution> ::= <DeviceType> <Capacity> *<Config>
<DeviceType> ::= Incubator | Autoclave | Centrifuge | ... [Known device
    types]
<Capacity> ::= <REAL> <MEAS>
<Config> ::= <ArgKey> <ArgValue>
<ArgKey> ::= Duration | Pace | Power | Quantity | ... [Known device
    configuration items]
<ArgValue> ::= <REAL> <MEAS>
```

D.3 Pre-processing of the protocols

The protocol pre-processing steps begin by reading all JSON files of the protocols. Each protocol is then splitted sentence-by-sentence using `Spacy`[1], with the constraint that every sentence is longer than ten characters. Due to the large volume of data, sentence splitting is handled in parallel. Afterwards, deeper sentence splitting is performed based on specific conditions for further refinement, such as the presence of `"and/then/and then"` followed by a verb[2]. We then parse sentences into root verbs and purpose clauses, which are identified using `token.dep_ == "ROOT"` for root verbs and `prepositional/adverbial/modals` for purpose clauses. Lastly, we merge phrases based on punctuation, and their classification into valid sentences or decorative phrases depends on whether they contain a root verb or lack a purpose clause.

The first verb in each sentence is extracted as an opcode, again utilizing parallel processing for efficiency. Opcode frequency is filtered to exclude stopwords, which are recorded in a separate text file. Then we categorize these opcodes into high-level operation classes using a GPT model (gpt-4o mini), where each opcode is classified into categories like `Transfer Operations`, `Transformation Operations`, or `Data Operations`.

Once operation classification is complete, entity recognition is performed (also using gpt-4o mini) to identify entities like `devices`, `input_flow_units`, `output_flow_units`, and `total_time`. Each flow unit is further categorized (also using gpt-4o mini) with a high-level classification composed of a phase, *i.e.*, `Gas`, `Liquid`, `Solid`, *etc.*; and a type, *i.e.*, `Chemical Compound`, `Biological Material`, *etc*. When both phase and type are successfully labeled, phase is preferred as the feature of the flow unit. If phase labeling fails, we use type the feature of the flow unit. If neither phase nor type is successfully labeled, the corresponding feature is set to `None`. Part of the rationale is that there are non-reagent components in the general sense, *i.e.*, `data`, `files`, `obscure or undefined substances`, *etc*. Therefore, we apply this strategy to maximize the possibility that there is a meaningful upper class labeling of the components without any redundancy.

Finally, we conduct a synonym merge process on the devices, which starts by using `transformers AutoTokenizer`[3] to get an embedding for each device name. Afterwards, we use `sklearn`[4] to identify potentially similar entity pairs by calculating the cosine similarity of the candidate entities, and then passing these entity pairs to the GPT model for synonym detection, thereby merging devices belonging to the same type. The reference names of these combined devices will be one of the features.

D.4 Pure LLM-based designer

The pure LLM-based designer employs RAG to retrieve similar protocols from the corresponding corpora for representation, following the design choice of the baseline in O'Donoghue et al. (2023). Specifically, in the `FB` approach, three similar protocols are first retrieved from the original protocol corpora using RAG, and then, along with the title and description of the target protocol, they are provided to the LLM to generate a NL plan. The LLM subsequently translates the NL plan into Python pseudocode. In the `IB` approach, three similar protocols' instance actions (like Python pseudofunctions definitions) are first retrieved from the corpora, and after randomizing their order, they are provided to the LLM along with the title and description of the target protocol to generate a plan in the form of Python pseudocode.

```
[Prompt for retrieving similar protocols from corpora]
You are an expert in biology and you are very familiar with the experiment protocols.
I would like to make a protocol for {title}.
I will give you some related protocols in the database.
Could you find me the most three similar and relevant protocols for reference in the given range?
```

[1] https://spacy.io/api/sentencizer

[2] https://spacy.io/api/matcher#_title

[3] https://huggingface.co/docs/transformers/v4.45.1/en/model_doc/auto#transformers.AutoTokenizer

[4] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html#cosine-similarity

```

Please output id of your selected protocols, separating with a comma. Don't output any other information.
[Output format]
id_1,id_2,id_3

[Related protocols]
{context}

Answer:
```

```
[Prompt for generating NL plan]
Your goal is to generate steps for a biology protocol.
These protocol steps must accurately describe a complete scientific protocol to obtain a result.
Steps of some similar protocols will be provided as a reference for you to generate the new one.
Output should only contain the steps without any other information.

Here is an example of how to generate steps for a biology protocol.

EXAMPLE:

{example protocol title}

Here are some extra details about the protocol:

{example protocol description}

example steps:

{example protocol steps}

YOUR TASK:
Generate steps for a protocol for {title}.

Here are some extra details about the protocol:

{details}

Here are some similar protocols' steps for reference:

{steps}

your steps:
```

```
[Prompt for translating NL plan to pseudocode]
Your goal is to convert biology protocols into python pseudocode.

EXAMPLE
```

```
Here is an example of how to convert a protocol for {example protocol title} into python pseudocode

{example protocol}

{example python pseudocode}

YOUR TASK:
Here is a biology protocol entitled '{title}' The protocol steps are as follows:

{protocol}

Please convert this protocol into python pseudocode.

python pseudocode:
```

```
[Prompt for generating plan in pseudocode]
Your goal is to generate python pseudocode for biology protocols.

Here is an example of how to generate pseudocode for a biology protocol.

EXAMPLE:

{example protocol title}

Here are some extra details about the protocol:

{example protocol description}

example pseudocode:

{example pseudocode}

YOUR TASK:
Generate pseudocode for a protocol for {title}.

Here are some extra details about the protocol:

{details}

You may only make use of the following python pseudocode functions:

{psuedofunctions}

your pseudocode:
```

D.5 Internal designer

The internal designer incorporates the specific representation as part of the prompt for an LLM, asking it to output the protocol while adhering to the given representation constraints, echoing the idea of Wang et al. (2023a). Specifically, in II, the instance actions retrieved from the corpora via RAG and the pseudofunctions definitions of the target protocol are shuffled and then provided together to the LLM, constraining it to generate a plan in the form of Python pseudocode using the given pseudofunctions definitions. In EI and EI+, relevant DSL instructions are selected from a domain-specific operation-centric view DSL and product-flow-centric view DSL, respectively. These instructions and the target protocol's title and description are provided to the LLM, prompting it to output the corresponding plan as instantiated DSL instructions.

```
[Protocol for generating plan in DSL program using operation-centric view DSL]
Your goal is to generate plan in domain specific language (DSL) for biology protocols.
The DSL specifications related to the operations involved in the experiment are provided. The DSL specification of each operation consists of multiple patterns, each pattern is an operation execution paradigm.

Here is an example of how to generate plan in DSL for a biology protocol.

EXAMPLE:

{example protocol title}

Here are some extra details about the protocol:

{example protocol description}

example plan in DSL:

{example plan}

[Requirements]
1. Design the experiment with finer granularity, incorporating more steps to complete the experiment in a more rigorous, complex, and comprehensive manner.
2. There are some missing parameters in the DSL specification. You should generate each step of the DSL program as detailed as possible based on your understanding of the protocol plan.
3. In Precond and Postcond, use formal name of the component to represent the SlotArg and EmitArg of each step. The component name should clearly describe the content of the component.

YOUR TASK:
Generate plan in DSL for a protocol for {title}.

Here are some extra details about the protocol:

{details}

You can choose to instantiate the following DSL specification to construct the DSL program:

```

```
{DSL}

Your plan in DSL program:
```

```
[Protocol for generating plan in DSL program using dual representation]
Your goal is to generate plan in domain specific language (DSL) for biology protocols.
Two perspectives of the DSL specification are provided: the specification for experimental operations and the specification for experimental products.
The DSL specification of each operation or product consists of multiple patterns, each pattern is an operation execution paradigm or a product flow paradigm.
Output every operation of the plan in the form of an operation DSL program and every product of the plan in the form of a product DSL program.

Here is an example of how to generate plan in DSL for a biology protocol.

EXAMPLE:

{example protocol title}

Here are some extra details about the protocol:

{example protocol description}

example plan in DSL:

{example plan}

YOUR TASK:
Generate plan in DSL for a protocol for {title}.

Here are some extra details about the protocol:

{details}

You can choose to instantiate the following DSL specifications to construct the DSL program:

Operation-view DSL specification:
{Operation-DSL}

Product-view DSL specification:
{Product-DSL}

Your plan in DSL program:
```

D.6 External designer

The external designer combines (i) deductive verification through DSL; and (ii) self-improvement by the LLM (Madaan et al., 2023). In EE, the external verifier is provided by the operation-centric view DSL and performs checks on two main aspects: (i) whether the precondition of each operation is an intermediate product of a previous step rather than appearing from nowhere; and (ii) whether the postcondition of each operation is used in subsequent steps rather than being omitted. Similarly, in EE+, the external verifier is provided by the DSL with a dual representation, focusing on cross-verifying the parallel dual tracks (the two perspectives of the DSL program). It checks whether the corresponding operation causes each *status transition* of the product: (i) whether the product in each product-view program is the output of its preceding operation; and (ii) whether the product in each product-view program is the input for its succeeding operation. If a mismatch occurs, the verifier generates corresponding error messages, such as *"Error: The product {product} required by operation {operation} at step {i} is not available from previous steps."* These error messages are then fed into the feedback-refine loop as feedback for the LLM to revise the plan. The loop terminates when the program passes the verification or reaches the maximum number of iterations, and the best result is retained based on the verification information.

```
[Prompt for refining the plan according to the feedback vertified by operation-centric view
DSL]
Your task is to improve a biology experimental protocol plan represented in domain-specific language (DSL) based on provided feedback.
The input plan in DSL consists of multiple DSL programs, each representing one step in the experimental protocol planning process, arranged in top-down order to indicate the execution sequence of operations.
Each DSL program has the following format:
{
    "Operation": ,    // Operation verb
    "Precond": {      // Precondition for this step
        "SlotArgNum": ,   // Number of arguments for the precondition
        "SlotArg":        // Input product for this step
    },
    "Execution": {
        "DeviceType": ,   // Execution device for the operation
        "Config": {       // dict of execution arguments - values
            Argkey: Argvalues
        }
    },
    "Postcond": {     // Postcondition for this step
        "EmitArgNum":,    // Number of arguments for the postcondition
        "EmitArg":        // Output product for this step
    }
}

The provided feedback indicates errors that occurred when compiling the DSL programs. You need to correct the program to ensure that the product is properly transferred between each step, i.e., the input product of each step must be the output from a previous step (except for the first step), and verify whether the output of each step is used as the input for subsequent steps (except for the final step).
```

```
If you believe the error in a particular step is due to the step preparing reagents rather than using a previous intermediate product, you can ignore this error.

Output your refined plan in DSL, returning a JSON block without any additional information or comments.

YOUR TASK:
Refine the plan in DSL for a protocol for {title}.

Here are some extra details about the protocol:

{details}

Refine the following plan:

{plan}

Here is the feedback of the plan:

{feedback}

Your refined plan in DSL:
```

```
[Prompt for refining the plan according to the feedback vertified by DSL with dual representation]
Your task is to improve a Biology experimental protocol plan represented in domain-specific language (DSL) based on provided feedback.
The input plan in DSL consists of multiple DSL programs from two perspectives: operation-view and product-view. The DSL programs from these two perspectives alternate and constrain each other.

This is the format of a product-view DSL program:
// Each product view DSL program represents the state of the product at that moment.
{
    Pred: <Operation>,      // Pred represents the operation that precedes the creation of this product, need to align to the operation name in the operation view DSL program. If the product is in its initial state, return "".
    FlowUnit: {      // FlowUnit defines the properties of the product being processed.
        Component: ,    // Component represents the actual product or material being processed, need to be the formal name of the component.
        ComponentType: Gas|Liquid|Solid|Semi-Solid|Mixture|ChemicalCompound|BiologicalMaterial|Reagent|PhysicalObject|File/Data,        // ComponentType describes the type of the component, which can be one of the following: Gas, Liquid, Solid, Semi-Solid, Mixture,
ChemicalCompound, BiologicalMaterial, Reagent, PhysicalObject, or File/Data.
        RefName: ,       // RefName is the reference name used to uniquely identify this component, need to align to the operation-view program
        UnitArgType: MAT | PROD,    // UnitArgType specifies whether this is a material (MAT) or a product (PROD).
        Vol: ,      // Vol represents the volume or quantity of the component.
        Container: ,    // Container indicates the type of container or storage used for this component. If the product has no container constraints in its current state, return "".
        Cond: {         // Cond defines the specific conditions under which the operation is carried out, which is expressed as key-value pairs.
            ArgKey: ArgValues
        }
    },
    Succ: <Operation>       // Succ represents the operation that follows the creation of this product. If the product is in its final state, return "".
}

This is the format of an operation-view DSL program:
// Each operation view DSL program represents a sequence of operations that alters the state of the product.
{
    Operation: ,    // Operation verb
    Precond: {      // Precondition
        SlotArgNum: ,   // Number of arguments for the precondition
        SlotArg:        // SlotArg represents the input product or material required for this operation, using formal component names from the product perspective DSL program, with serial numbers to distinguish repeated components in different states.
    },
    Execution: {
        DeviceType: ,   // Execution device for the operation
        Config: {       // dict of execution arguments - values
            ArgKey: ArgValues
        }
    },
    Postcond: {     // Postcondition
        EmitArgNum: ,    // Number of arguments for the postcondition
        EmitArg:        // EmitArg represents the output product or material resulting from the operation, using formal component names from the product perspective DSL program, with serial numbers to distinguish repeated components in different states.
    }
}

```

```
The provided feedback indicates errors that occurred when compiling the DSL programs. You need to correct the program to ensure that the state changes of each product's RefName in the Product-view are caused by the corresponding operations in the Operation-view.
If you believe the error in a particular step is due to a mismatch in product names between the two perspectives rather than an actual error, you can ignore this error.

Output your refined plan in DSL, returning a JSON block without any additional information or comments.

YOUR TASK:
Refine the plan in DSL for a protocol for {title}.

Here are some extra details about the protocol:

{details}

Refine the following plan:

{plan}

Here is the feedback of the plan:

{feedback}

Your refined plan in DSL:
```

D.7 Computing load of the machine designers

For automated representation generation, we primarily used GPT-4o mini with OpenAI's Batch API[5] for preprocessing, incurring a cost of approximately $60 across four domains. The design of the DSLs was executed on a MacBook with an M2 chip, running 1,000 iterations to ensure convergence. This process required an average of 55 seconds per iteration for the operation-centric view DSL and an average of 2 seconds per iteration for the product-centric view DSL. For the machine designer, we primarily utilized GPT-4o mini combined with RAG for design, with a total cost of approximately $10 (7 methods, 140 protocols). In summary, the overall computational load is relatively low, highlighting the accessibility of our machine designers when utilizing the proposed representations and the corresponding automatic representation generation modules.

E Data collection

E.1 Corpora sources

The corpora $\mathcal{C}$ for the automatic generation of representations (Sec. 3.1) and the corpora for selecting the testing set (Sec. 4.1) are both retrieved from open-sourced websites run by top-tier publishers, including Nature's Protocolexchange[6], Cell's Star-protocols[7], Bio-protocol[8], Wiley's Current Pro-

[5] https://platform.openai.com/docs/guides/batch/batch-api

[6] https://protocolexchange.researchsquare.com/

[7] https://star-protocols.cell.com/

[8] https://bio-protocol.org/en

tocols[9], and Jove[10]. These sources compile a dataset of 15,837 experimental protocols across four domains: Genetics (8794 protocols), Medical (7351), Ecology (812), and Bioengineering (3597), with minimal overlap between them. We aggregated the corpora and analyzed the themes of the protocols according to the first- and second-level labels attached to them. We adopt measures to ensure that $\mathcal{C}$ is mutually exclusive with the testing set.

Other domains, such as Physics and Chemistry, are also representative domains of experimental sciences, besides Biology, Medical, and Ecology. The preliminary factor that restricts our current scope is data accessibility. Due to the higher cost of accessing the corpora of protocols for conducting physics and chemistry experiments, for example, mining the protocol from the "method" section of relevant published papers, we leave the application to Physics and Chemistry for future work.

E.2 Eliminating the risk of data leaking

We employ the broadly accepted standard operating process to empirically verify that LLMs have not memorized the data we use. We adopt the methodology outlined in Section 5.2 of *Skywork* (Wei et al., 2023) and draw upon recent studies on detecting memorization in LLMs (Carlini et al., 2021; 2022). Specifically, we use gpt-4o mini to synthesize data resembling the style of steps from novel protocols, and then calculate the perplexity on the test set and reference set. Since the reference set is newly generated, we consider it clean, not belonging to any training set of any model.

We randomly sample 100 sequences each from the test set and the reference set of the novel protocols. Each sequence corresponds to a single procedural step described in NL. We truncate the final 50 tokens of each sequence, retaining the prefixes. These prefixes are then used as prompts for the LLM to predict the next 50 tokens, for which we calculate the perplexity. If the perplexity of the test set is significantly lower than that of the reference set, the test set might have appeared in the model's training phase.

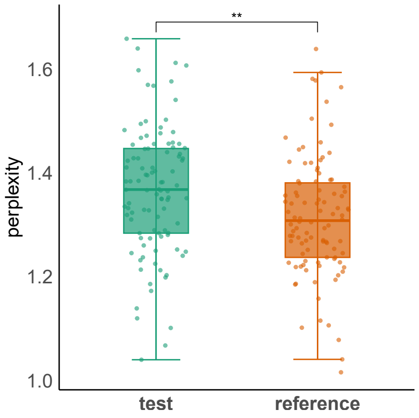


Figure A1: **Comparison between the perplexity of the test set and the reference set**

The results indicate that the LLM's average perplexity on the test set is significantly higher than that on the reference set ($t(198) = 3.040, \mu_d < 0, p < .05$; see Fig. A1), suggesting that the LLM encounters greater uncertainty with the novel protocols in the test set. This finding implies that for a published, widely accepted, and standardized operating process, there is no evidence to suggest that the LLM has memorized the data.

E.3 On the diversity of novel protocols

Assessing diversity among novel protocols is both informative and meaningful. To further support our analysis, we incorporate a t-SNE visualization of the experimental objectives (described in natural language) for the novel protocols we select, as shown at Fig. A2. The results demonstrate a well-dispersed distribution, indicating a sufficient level of diversity among the protocols.

E.4 Showcases

```
[Protocol 1 - Bioengineering]
Preparation of lysates
1. Harvest approximately 1 x 10^7 cells by centrifugation at 2000 RPM for
    5 min. Aspirate media and resuspend cell pellet with 1 mL of ice-
   cold PBS and transfer to a 1 mL centrifuge tube. Microcentrifuge at
   2000 RPM for 5 min at 4 °C.
```

[9] https://currentprotocols.onlinelibrary.wiley.com/

[10] https://www.jove.com/

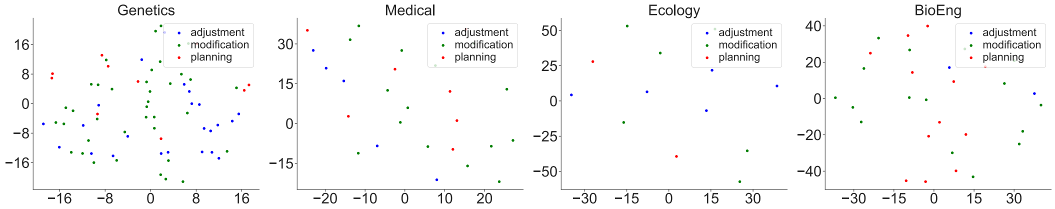


Figure A2: **Visualization of diversity between novel protocols**

```
2. Aspirate PBS, and then add Hypotonic Buffer (supplemented with 1%
    Triton X-100, to disrupt membrane and cytoskeleton-bound MEKK1
    fractions).
3. Cell lysates are homogenized by passing through 22-gauge needles, and
    tubes are put on ice for 15 min to complete the lysis. Crude extracts
     are then centrifuged at 2500 RPM for 5 min. Supernatants are
    transferred to fresh centrifuge tubes, and cold 5 M NaCl is added to
    each sample to make a salt concentration of between 0.7-1.0 M to
    disrupt protein-protein interactions.
4. Spin the crude extracts by ultracentrifugation at 55000 RPM to
    properly pellet residual insoluble proteins from the extract.
    Transfer supernatants into fresh centrifuge tubes.
Immunoprecipitation
5. Rinse Protein A beads in Hypotonic Buffer and place on ice until ready
     for use.
6. Take a volume of cell lysates (prepared as described above), and
    dilute with Hypotonic Buffer to 250-500 mM salt to enable protein-
    protein interactions.
7. Add 2 μg of preclearing antibody to the diluted lysate (e.g., anti-Myc
     or anti-p65), vortex, add 50 μL of Protein A beads, and rock for 45
    min.
8. Touchspin samples, and transfer supernatant to a fresh tube.
9. Add 2 μg of polyclonal anti-MEKK1 to the lysates, and rock for 1 h.
    After this period, add 50 μL of Protein A beads and rock tubes at 4 °
    C for 1 h.
10. Touchspin beads, wash beads with hypotonic buffer (supplemented with
    NaCl to a concentration of 300 mM), vortex, and rock for 10 min. In
    total, 3-5 washes of the beads are performed.
11. Finally, wash once with Hypotonic Buffer, and resuspend in Kinase
    Assay Buffer. Purified MEKK1 may be stored by snap-freezing in liquid
     nitrogen and long-term storage at -80 °C. Kinase assay Following
    preparation of MEKK1 immunoprecipitates (as above), incubate with 7 μ
    g of JNKK1(K131M) along with 5 μCi of ATP in Kinase Assay Buffer for
    30 min at 30 °C."

[Protocol 2 - Genetics]
1. Note that everything is in DEPC water. Inoculate W303a cells
    expressing different TOR1-RR variants in 2 mL SC medium overnight.
2. Subculture the cells starting from OD600=0.1 in 10 mL SC media, shake
    vigorously at 30 °C, 300 RPM for around 4-6 h until OD600=0.4-0.5.
```

```
3. Collect the cells by spinning down without freezing on ice. Discard
    supernatant.
4. Re-suspend cells with 1 mL water and transfer to a 1.5 eppendorf tube,
     quickly spin down at 3,000 x g for 15 sec.
5. Re-suspend cell pellet in 400 μL of AE buffer at room temperature.
6. Add 40 μL 10% SDS (final around 1%) and vortex briefly at room
    temperature (RT).
7. Immediately add 500 μL hot phenol/AE (put in 65 °C for 10 min before
    use), vortex vigorously for 1 min. Incubate at 65 °C for 5 min.
    Briefly vortex every 30 sec.
8. Immediately freeze by dumping into liquid nitrogen. Wait to thaw at RT
     (put in 30 °C to thaw may crack the tube).
9. Centrifuge for 10 min on a standard laboratory microfuge at 20,000 x g
     at RT.
10. Transfer around 400 μL supernatant to a new eppendorf tube. Recycle
    the lower phenol fraction carefully following the chemical safety
    protocol in your laboratory.
11. Add equal volume (400 μL) phenol: CHCl3/AE-Na. Vortex vigorously for
    1 min at RT.
12. Spin down at 20,000 x g for 5 min in a standard laboratory microfuge.
13. Transfer supernatant (around 350 μL) to a fresh 1.5 mL eppendorf tube
    . Add CHCl3: isoamyl alcohol (24:1). Vortex vigorously for 1 min at
    RT.
14. Transfer aqueous supernatant to a fresh 1.5 mL microfuge tube. If
    white cloudy precipitate is observed between the aqueous phase and
    organic phase, repeat steps 17-18.
15. Add 1/10 volume of 3 M NaOAc (pH 5) and vortex vigorously. Add 2.5
    volumes of ethanol. Vortex again.
16. Place at -20 °C for at least 30 min.
17. Spin down in the microfuge at 20,000 x g, 15 min at 4 °C. RNA pellet
    is usually visible.
18. Add ice-cold 75% EtOH, place at 4 °C for around 10 min. Vortex and
    spin down on microfuge 20,000 x g, 15 min at 4 °C. Discard
    supernatant. Suck out the liquid droplets in the tube. The white RNA
    pellet will turn clear when it dries out. Add 30-50 μL ddH2O (DEPC)
    immediately after it becomes clear. Do not let the RNA over-dry,
    which will make it difficult to dissolve. If RNA pellet is over-dry,
    dissolve RNA at 37 °C for 30 min. Store RNAs at -80 °C for more than
    2 months."

[Protocol 3 - Medical]
1. Passage through a 45 μm filter. Add 100 μL/well of 100 μg/mL salmon
    sperm DNA to a 96-well Microtest assay plate.
2. Wrap the plate with plastic wrap and incubate at 4 °C overnight.
3. Discard the coating antibody solution and wash the plate with 1x PBS-
    Tween 6 times.
4. Dry the plate and add 100 μL of blocking solution per well to the
    plate.
5. Incubate the plate at room temperature (RT) for 1.5 h.
```

```
6. Discard the blocking solution and wash the plate with 1x PBS-Tween 5
    times.
7. Dry the plate and keep it at 4 °C for later use.
8. Harvest the spleen and create a single-cell suspension by gently
    smashing spleen pieces with the frosted surface of a pair of
    microscope slides in 5 mL of DMEM.
9. Transfer the cells into 50 mL conical tubes and spin down the cells at
     300 RCF for 5 min at 4 °C.
10. Discard the supernatant with aspiration without disturbing the pellet
    .
11. Re-suspend the cells with 5 mL of 0.17 M ammonium chloride and keep
    the cells on ice for 5 min.
12. Add 15 mL DMEM to the cells and spin at 300 RCF for 5 min at 4 °C.
13. Discard the supernatant and re-suspend the cells with 20 mL of DMEM
    and count the cells.
14. Re-suspend 2 x 10^7 cells in 2 mL of 10% DMEM and make a three-fold
    serial dilution (a total of 8 dilutions) with 10% DMEM.
15. Add 50 µL/well of the serial dilutions on the DNA-coated plate and
    centrifuge at 300 RCF for 5 min at 4 °C.
16. Incubate the cells at 30 °C for 2 h in a cell-culture incubator with
    6% CO2.
17. Add 50 µL/well of biotin-conjugated anti-IgM or anti-IgG (1:350 in
    10% DMEM) to the cells.
18. Centrifuge the cells at 300 RCF for 5 min at 4 °C and incubate the
    cells overnight in a cell-culture incubator with 6% CO2.
19. Discard the cells and wash the plates 10 times with 10x PBS-Tween 20.
20. Dry the plates and add 50 µL of streptavidin alkaline phosphatase
    (1:1,000 in 1% BSA/PBS) to the plate.
21. Incubate the plate at RT for 1 h and wash the plate 10 times with 10x
     PBS-Tween 20.
22. Dry the plate and add 50 µL/well of 1 mg/mL BCIP in AMP buffer to
    develop the plate.
23. When the spots are clearly visible under a dissecting microscope,
    stop the development by discarding the BCIP solution and rinsing the
    plate with tap water thoroughly.
24. Spots can be counted using a dissecting microscope or using an
    ELISpot reader."
```

F Reproducibility

The project page with supplementary files for reproducing the results of this paper will be available at `https://autodsl.org/procedure/papers/iclr25shi.html`.

G Limitations

As a representation designed for a relatively new problem, the design and evaluation of the proposed framework come with limitations, leading to further investigations:

- Overall, our method achieves promising results across the four domains. Specifically, it performs best in experimental design for Genetics, shows comparable effectiveness in Medical and Bioengineering, but is less effective in Ecology. Notably, the Genetics corpus is the largest among the four domains, while the Ecology corpus is significantly smaller than the others. These observations suggest a potential positive correlation between the size of the domain-specific corpus and the “quality” of the resulting DSL. In other words, a larger corpus may lead to a “better” representation, thereby influencing the outcomes of protocol design. This hypothesis necessitates further investigation through rigorously designed experiments and carefully defined metrics for evaluating what constitutes a “better” representation.

- We majorly consider the imperative programming DSLs as the implementation of representation in this work. This raises the question of whether incorporating objective-oriented programming paradigms could enhance the representation of complex entities within protocols, particularly the properties of reagents and intermediate products. If we are able to make the DSLs model the fine-grained reactions between different components and automate the design of those DSLs based on a broader source of data, such as the Wikipedia pages, we can ultimately manage to build up a symbolic digital twin for a domain-specific system, such as the cell cultivation environment. Such simulation systems may greatly benefit protocol design with their power of prediction, explanation, and counterfactual analysis.
- Can we explicitly extend our proposed representation to a hierarchical graph, thereby establishing the foundation for employing the advanced algorithms on graph routing and graph optimization? Results on the hierarchical graph can also serve as a external heuristic and constraint for LLM-based protocol designers. This hybrid approach may combine both the advantages of LLMs, *i.e.*, exploitation of background knowledge, and those of classical algorithms, *i.e.*, white-boxed properties with high explainability.
- Can we apply the representation and the automatic representation generator to other critical domains with a high demand for automating procedure design, such as designing product route sheets for advanced manufacturing?

With many questions unanswered, we hope to explore more on automated protocol design for self-driving laboratories and beyond.

H THE AUTOMATICALLY GENERATED REPRESENTATIONS

H.1 OPERATION-CENTRIC VIEW DSL

```
{
    "Operation": "Precipitate",
    "pattern_0": {
        "Precond": {
            "SlotArgNum": 2,
            "SlotArg": ["Liquid", "Solid"]
        },
        "Execution": {
                "DeviceType": "falcon tube",
                "Capacity": "15 mL",
                "Config": {}
        },
        "Postcond": {
            "EmitArgNum": 1,
            "EmitArg": ["Liquid"]
        },
        "Example": [
            "Precipitate RNA by adding 600 μL of 100 % EtOH, 20 μL of 3 M NaOAc ( pH 5.5 ), and 3 μL of glycogen .",
            "Precipitate the DNA in each tube by adding 20 μl of 3 M sodium acetate ( pH 5.2 ) and 550 μl of 100 % ethanol .",
            "Ethanol precipitate the RNA by adding 5 μl 3 M sodium acetate ( pH 5.2 ) ,2 μl of glycogen ( 20 mg / ml ) ,and 171 μl of 100 % ethanol .",
        ]
    },
    "pattern_1": {
        "Precond": {
            "SlotArgNum": 4,
            "SlotArg": ["Liquid", "Liquid", "Liquid", "Liquid"]
        },
        "Execution": {
                "DeviceType": "centrifuge",
                "capacity": "1.5 ml",
                "Config": {
                    "time": "10-15 min",
                    "speed": ["12,000 × g", "20,000 × g"],
                    "temperature": "4 °C",
                }
        },
        "Postcond": {
            "EmitArgNum": 1,
            "EmitArg": ["Solid"]
        },
        "examples": [
            "Precipitate the cell debris in the lysate by centrifugation at 20,000 × g for 10-15 min at 4 ° C.",
            "precipitate DNA with 13.5 µL of following mixture (1 µL of 20 mg / ml Glycogen, 12.5 µL of 3 M NaOAc [pH 5.3]) and 340 µL ethanol.",
            "precipitate the total RNA by centrifuging at 12,000 × g for 15 min at 4 ° C."
        ]
    }
},
{
    "Operation": "Spin",
    "pattern_0": {
        "Precond": {
            "SlotArgNum": 2,
            "SlotArg": ["Liquid", "Liquid"]
        },
        "Execution": {
            "DeviceType": "spin plate",
            "Config": {
                "time": ["1 min"]
            }
        },
        "Postcond": {
            "EmitArgNum": 1,
            "EmitArg": ["Physical Object"]
        },
        "Example": [
            "Nuclei washing and tagmentation: Spin down nuclei at 600 g for 10 mins at 4 ° C , resuspended with 50 μL Complete Buffer.",
            "Spin the sample at 4,000 × g at 4 ° C until the volume reduces to about 1 mL. Quantify protein concentration as described in step 60.",
            "Spin down the 15 mL tubes at 2,500 ×g and 4 ° C for 20 min.",
            "Spin for 2 min at 1,000 x g. Save a few μL of concentrated sample to run on an agarose gel later.",
            "Spin the tube for 30 sec at 12,000 x g to consolidate the gel at the bottom of the tube.",
            "Spin plate at 300×g for 1 min to collect liquid at the bottom of the wells.",
            "Once NXT PCR program is complete, quick spin the sample tube then place it on the magnet for 1 min. Transfer the supernatant containing the amplified mRNA-seq library into a new PCR tube.",
            "Spin down 2 mill nuclei at 600×g for 5 min (whole liver nuclei) or use a magnet (bead-bound nuclei)."
        ]
    },
    "pattern_1": {
        "Precond": {
            "SlotArgNum": 1,
            "SlotArg": ["Mixture"]
        },
        "Execution": {
            "DeviceType": "microcentrifuge",
            "Config": {}
        },
        "Postcond": {},
        "Example": [
            "Small volumes, 1-3 mL should be spun in a small tube where these fewer EVPs can more readily be collected.",
            "Briefly spin down the bead-lysate mixture.",
            "Spin down the mix tube to eliminate bubbles/air in a bench microcentrifuge. Add 19 μL of the mix to each well."
        ]
    },
    "pattern_2": {
        "Precond": {
            "SlotArgNum": 1,
            "SlotArg": ["Liquid"]
        },
        "Execution": {
            "DeviceType": "centrifuge",
            "Config": {
                "speed": ["800 g"],
                "time": ["7 min"]
            }
        },
        "Postcond": {
            "EmitArgNum": 1,
            "EmitArg": ["Liquid"]
        },
        "Example": [
            "Spin lysate at 14 krcf for 10 min at 4 ° C; transfer cleared lysate to new tube.",
            "Spin down the beads for 60 s at 2,000 x g. Discard the supernatant by carefully pipetting out the buffer.",
            "Spin at 12,000 × g until the total volume in both filters is reduced to 120 μL (<=30 min). Keep aside 5 μL of purified labeled histone for SDS-PAGE analysis.",
            "Quickly spin the FACS tube to allow the cell suspension to pass through the filter to remove undigested large tissue debris.",
            "Spin once for 7 min at 800 g. Use the BD cytofix/cytoperm kit according to the manufacturer's instructions and thereafter add antibodies for intracellular detection of IFN and TNF."
        ]
    }
},
{
    "Operation": "Sonicate",
    "pattern_0": {
        "Precond": {
            "SlotArgNum": 1,
            "SlotArg": ["Liquid"]
        },
        "Execution": {
            "DeviceType": "sonicator",
            "Config": {
                "time": ["20 - 30 s"]
            }
        },
        "Postcond": {
            "EmitArgNum": 1,
            "EmitArg": ["Semi-Solid"]
        },
        "Example": [
            "Sonicate the pellet suspension on ice under a 50 % duty cycle for 5 min.",
            "Agarose gel of sonicated Arabidopsis chromatin.",
            "Sonicate proteoliposomes for 20 - 30 s or 3 times for 10 s, placing on ice in between sonication, if necessary.",
            "The lipid suspension is sonicated to form small unilamellar vesicles (SUVs)."
        ]
    },
    "pattern_1": {
        "Precond": {
            "SlotArgNum": 2,
            "SlotArg": ["Liquid", "Solid"]
        },
        "Execution": [
            {
                "DeviceType": "bransonic",
                "Config": {
                    "temperature": ["60 ° C"],
                    "time": ["90 min"]
                }
            },
            {
                "DeviceType": "sonicator",
                "Config": {}
            }
        ],
        "Postcond": {},
        "Example": [
            "Sonicate 10 μg BAC DNA or 50 μg genomic DNA in total (you will recover 10 % DNA after sonication and size selection).",
            "Sonicated chromatin is immunoprecipitated with the chosen antibodies and non-enriched chromatin washed with a series of washing buffers.",
            "If the herring sperm DNA has not been sufficiently sonicated or too much has been used, the DNA pellet might not adhere to the microfuge tube and can be lost with the ethanol.",
            "Sonicate the lipid tube to dissolve lipids with the mineral oil for 90 min at 60 ° C by using Bransonic."
        ]
    }
}
```

H.2 PRODUCT-FLOW-CENTRIC VIEW DSL

```
{
    "Pred": "Modification Operations",
    "FlowUnit": {
        "Component": "FBS",
        "ComponentType": "Liquid",
        "UnitArgType": "MAT"
        "Vol": ["0.1 mL", "0.5 mL", "1 mL", "1.5 mL", "2 mL", "3 mL", "5 mL", "10 mL", "25 mL", "50 mL", "400 μL", "500 μL", "500 mL"],
        "Container": "Tube",
        "Cond": {
            "Concentration": ["0.5%", "1%", "2%", "2.5%", "5%", "10%", "15%", "20%", "30%", "50%", "90%", "100%"],
            "Temperature": ["-150°C", "4°C", "18°C-26°C", "37°C", "56°C"],
            "State": "heat-inactivated"
        }
    "Succ": "Transfer Operations"
},
{
    "Pred": "Detection and Measurement Operations",
    "FlowUnit": {
        "Component": "ethidium bromide",
        "ComponentType": "Solid",
        "UnitArgType": "MAT",
        "Vol": ["0.25 μL/mL", "0.5 μL/mL", "2 - 3 μL", "10 μL", "15 μL", "10 μg/mL", "0.5 μg/μL"],
        "Container": "Flask",
        "Cond": {
            "Concentration": ["0.0024%", "0.3 - 10 μg/mL", "1.5% (w/v)", "5 μM", "1:1000"],
            "Temperature": ["25°C", "room temperature"],
            "State": ["toxic", "carcinogenic"],
            "Charge": ["positively charged"]
        }
    },
    "Succ": "Modification Operations"
},
{
    "Pred": "Transfer Operations",
    "FlowUnit": {
        "Component": "gel",
        "ComponentType": "Semi-Solid",
        "UnitArgType": "MAT",
        "Vol": ["0.5 mL"],
        "Container": ["Gel Cassette", "Tank", "Tube"],
        "Cond": {
            "Impedance": ["under 20 kOhm"],
            "Size": ["50-250 nt"]
        }
    },
    "Succ": "Transfer Operations"
}
```

I CASE STUDIES

I.1 CASE STUDY: CONTRIBUTIONS OF THE BUILDING BLOCKS

Part of protocol designed by EE+:

```
{
    "Pred": "",
    "FlowUnit": {
        "Component": "Lysis solution",
        "ComponentType": "Liquid",
        "RefName": "Lysis_solution-1",
        "UnitArgType": "MAT",
        "Vol": "50 μL",
        "Container": "",
        "Cond": {
            "State": "Liquid"
        }
    },
    "Succ": "Pipette"
},

{
    "Operation": "Pipette",
    "Precond": {
        "SlotArgNum": 1,
        "SlotArg": [
            "Lysis_solution-1"
        ]
    },
    "Execution": {
        "DeviceType": "Pipette",
        "Config": {
            "time": "10 times",
            "volume": "50 μL"
        }
    },
    "Postcond": {
        "EmitArgNum": 1,
        "EmitArg": [
            "Lysis_solution-2"
        ]
    }
}
```

Part of protocol designed by EE:

```
{
    "Operation": "Add",
    "Precond": {
        "SlotArgNum": 1,
        "SlotArg": [
            "Triton-X"
        ]
    },
    "Execution": {
        "DeviceType": "8-channel pipette",
        "Config": {
            "Volume": "1% solution"
        }
    },
    "Postcond": {
        "EmitArgNum": 1,
        "EmitArg": [
            "Triton_X_Solution"
        ]
    }
}

{
    ”Operation”: ”Run”,
    ”Precond”: {
        ”SlotArgNum”: 1,
        ”SlotArg”: [
            ”Cell_Lysis_Mixture”
        ]
    },
    ”Execution”: {
        ”DeviceType”: ”Thermal Cycler”,
        ”Config”: {
            ”Temperature”: ”70 °C”,
            ”Time”: ”15 min”
        }
    },
    ”Postcond”: {
        ”EmitArgNum”: 1,
        ”EmitArg”: [
            ”cDNA_Reverse_Transcription”
        ]
    }
}
```

Part of protocol designed by EI:

```
{
    ”Operation”: ”Run”,
    ”Precond”: {
        ”SlotArgNum”: 1,
        ”SlotArg”: [
            ”Cell_Lysis_Mixture”
        ]
    },
    ”Execution”: {
        ”DeviceType”: ”Thermal Cycler”,
        ”Config”: {
            ”Temperature”: ”70 °C”,
            ”Time”: ”15 min”
        }
    },
    ”Postcond”: {
        ”EmitArgNum”: 1,
        ”EmitArg”: [
            ”cDNA_Reverse_Transcription”
        ]
    }
}
```

Part of protocol designed by II:

```
"reverse_transcribe": {
    "muscs": "cells",
    "buffer": "RT buffer",
    "enzyme": "reverse transcriptase",
    "incubation_time": "60 minutes",
    "temperature": "42"
}

"prepare_single_cell_suspension": {
    "input_cells": "lysed cells"
}
```

Part of protocol designed by IB:

```
"reverse_transcribe": {
    "muscs": "RNA",
    "buffer": "reverse transcription buffer",
    "enzyme": "reverse transcriptase",
    "incubation_time": "60 minutes",
    "temperature": "42"
}

"prepare_single_cell_suspension": {
    "input_cells": "single-cell samples"
}
```

Part of protocol designed by FB:

```
"sort_single_cell": {
    "plate": "PCR plate",
    "nozzle_size": "100 μm",
    "mode": "single-cell purity"
}
```

I.2 Case study: handling different task complexities

Part of protocol designed in Planning:

```
{
    "Operation": "Obtain",
    "Precond": {
        "SlotArgNum": 1,
        "SlotArg": [
            "File/Data"
        ]
    },
    "Execution": {
        "DeviceType": "QIAGEN Blood & Cell Culture DNA Maxi Kit",
        "Config": {}
    },
    "Postcond": {
        "EmitArgNum": 1,
        "EmitArg": [
            "HMW genomic DNA"
        ]
    }
}
```

Part of protocol designed in Modification:

```
{
    "Operation": "Centrifuge",
    "Precond": {
        "SlotArgNum": 1,
        "SlotArg": [
            "Serum_Plasma_in_PBS-1"
        ]
    },
    "Execution": {
        "DeviceType": "Ultracentrifuge",
        "Config": {
            "speed": [
                "12,000 × g"
            ],
            "time": [
                "20 min"
            ],
            "temperature": [
                "4 °C"
            ]
        }
    },
    "Postcond": {
        "EmitArgNum": 1,
        "EmitArg": [
            "Pellet-1"
        ]
    }
}
```

Part of protocol designed in Adjustment:

```
{
    "Operation": "Incubate",
    "Precond": {
        "SlotArgNum": 2,
        "SlotArg": [
            "Washed sections with 1st antibody-1",
            "2nd antibody mixture-1"
        ]
    },
    "Execution": {
        "DeviceType": "Moistening box",
        "Config": {
            "temperature": "37 C",
            "time": "1 h"
        }
    },
    "Postcond": {
        "EmitArgNum": 1,
        "EmitArg": [
            "Sections with 2nd antibody-1"
        ]
    }
}
```

I.3 CASE STUDY: GENERALITY ACROSS DOMAINS

Part of protocol designed for Bioengineering:

```
{
        {
            "Operation": "Incubate",
            "Precond": {
                "SlotArgNum": 2,
                "SlotArg": [
                    "Lysis Mixture-2",
                    "Stop Buffer-1"
                ]
            },
            "Execution": {
                "DeviceType": "Thermocycler",
                "Config": {
                    "temperature": "65 °C",
                    "time": "30 min"
                }
            },
            "Postcond": {
                "EmitArgNum": 1,
                "EmitArg": [
                    "Neutralized Mixture-1"
                ]
            }
        },
        {
            "Pred": "Incubate",
            "FlowUnit": {
                "Component": "Neutralized Mixture",
                "ComponentType": "Mixture",
                "RefName": "Neutralized Mixture-1",
                "UnitArgType": "PROD",
                "Vol": "60 μL",
                "Container": "0.2-ml PCR tube",
                "Cond": {
                    "State": "Neutralized"
                }
            },
            "Succ": "Mix"
        },
        {
            "Operation": "Elute",
            "Precond": {
                "SlotArgNum": 1,
                "SlotArg": [
                    "Washed Beads-1"
                ]
            },
            "Execution": {
                "DeviceType": "Centrifuge",
                "Config": {
                    "time": "1 min"
                }
            },
            "Postcond": {
                "EmitArgNum": 1,
                "EmitArg": [
                    "Eluted Product-1"
                ]
            }
        },
        {
            "Pred": "Elute",
            "FlowUnit": {
                "Component": "Eluted Product",
                "ComponentType": "BiologicalMaterial",
                "RefName": "Eluted Product-1",
                "UnitArgType": "PROD",
                "Vol": "50 μL",
                "Container": "0.2-ml PCR tube",
                "Cond": {
                    "State": "Eluted"
                }
            },
            "Succ": "Incubate Clear"
        },
        {
            "Operation": "Quantify",
            "Precond": {
                "SlotArgNum": 1,
                "SlotArg": [
                    "Clear Eluted Solution-1"
                ]
            },
            "Execution": {
                "DeviceType": "Nanodrop",
                "Config": {}
            },
            "Postcond": {
                "EmitArgNum": 1,
                "EmitArg": [
                    "Quantified Sample-1"
                ]
            }
        },
        {
            "Pred": "Quantify",
            "FlowUnit": {
                "Component": "Quantified Sample",
                "ComponentType": "Liquid",
                "RefName": "Quantified Sample-1",
                "UnitArgType": "PROD",
                "Vol": "50 μL",
                "Container": "0.2-ml PCR tube",
                "Cond": {
                    "State": "Quantified",
                    "Concentration": "150 ng/μL",
                    "A260/A280": 1.85,
                    "A260/A230": 2.1
                }
            },
            "Succ": "Dilute"
        }
}
```

Part of protocol designed for Ecology:

```
{
        {
            "Operation": "Grow",
            "Precond": {
                "SlotArgNum": 1,
                "SlotArg": [
                    "Watered_Rice_Plants-1"
                ]
            },
            "Execution": {
                "DeviceType": "Environmental growth chamber",
                "Config": {
                    "Temperature": "24 °C",
                    "LightCycle": "12h light/12h dark"
                }
            },
            "Postcond": {
                "EmitArgNum": 1,
                "EmitArg": [
                    "Mature rice plants"
                ]
            }
        },
        {
            "Pred": "Grow",
            "FlowUnit": {
                "Component": "Mature rice plants",
                "ComponentType": "BiologicalMaterial",
                "RefName": "Mature_Rice_Plants-1",
                "UnitArgType": "PROD",
                "Vol": "N/A",
                "Container": "Plastic pot",
                "Cond": {
                    "State": "Mature",
                    "Height": "50-60 cm"
                }
            },
            "Succ": "Anesthetize"
        },
        {
            "Operation": "Collect",
            "Precond": {
                "SlotArgNum": 2,
                "SlotArg": [
                    "Monitored_Aphid-1",
                    "Mature_Rice_Plants-1"
                ]
            },
            "Execution": {
                "DeviceType": "Microcapillary tube",
                "Config": {}
            },
            "Postcond": {
                "EmitArgNum": 1,
                "EmitArg": [
                    "Phloem sap"
                ]
            }
        },
        {
            "Pred": "Collect",
            "FlowUnit": {
                "Component": "Phloem sap",
                "ComponentType": "Liquid",
                "RefName": "Phloem_Sap-1",
                "UnitArgType": "PROD",
                "Vol": "1-2 μL",
                "Container": "Microcapillary tube",
                "Cond": {
                    "State": "Collected",
                    "Appearance": "Clear, slightly viscous"
                }
            },
            "Succ": "Dilute"
        },
        {
            "Operation": "Centrifuge",
            "Precond": {
                "SlotArgNum": 1,
                "SlotArg": [
                    "Diluted_Phl_Sap-1"
                ]
            },
            "Execution": {
                "DeviceType": "Centrifuge",
                "Config": {
                    "speed": "6000 rpm",
                    "temperature": "4 °C",
                    "time": "10 min"
                }
            },
            "Postcond": {
                "EmitArgNum": 1,
                "EmitArg": [
                    "Extracellular vesicles"
                ]
            }
        },
        {
            "Pred": "Centrifuge",
            "FlowUnit": {
                "Component": "Extracellular vesicles",
                "ComponentType": "Mixture",
                "RefName": "EVs-1",
                "UnitArgType": "PROD",
                "Vol": "N/A",
                "Container": "200-µl microtube",
                "Cond": {
                    "State": "Purified",
                    "Appearance": "Small, almost invisible pellet"
                }
            },
            "Succ": ""
        }
}
```