

ProtoAct: Turning Wet-Lab Protocols into Embodied Robotic Actions

Zhe Liu^{1,2*}, Jiaming Gu^{1,2*}, Zhaohui Du^{1,2*}, Zhe Wang^{1,2†},
Huanbo Jin^{1,2}, Quan Lu^{1,2}, Qi Wang³, Ting Xiao^{1,2},
Minting Pan⁴, Dongzhan Zhou⁴

¹Key Laboratory of Smart Manufacturing in Energy Chemical Process, Ministry of Education,
East China University of Science and Technology, Shanghai, China

²Department of Computer Science and Engineering,
East China University of Science and Technology, Shanghai, China

³Department of Laboratory Medicine, Ruijin Hospital,
Shanghai Jiao Tong University School of Medicine, Shanghai, China

⁴AI for Science Center, Shanghai AI Laboratory, Shanghai, CN
wangzhe@ecust.edu.cn

Abstract

Biological wet-lab protocols are written for trained researchers and often leave routine operations, state-dependent conditions, and contextual parameters implicit, making them difficult to translate into robot-executable actions. We present ProtoAct, a structured protocol-grounding framework that converts free-form biological procedures into state-aware, embodiment-ready action sequences. ProtoAct uses ProtoRAG to retrieve manually annotated examples for context-sensitive parsing, employs RefineChecker to detect and revise missing or inconsistent steps, and applies ActSchema to map the refined procedure into constrained JSON function sequences. We further introduce BioP2E, for which we manually annotate 22 cell-culture protocols into 258 monitoring conditions, 910 executable subtasks, and 962 grounded action calls. Evaluation across seven large language models demonstrates that ProtoAct can be effectively instantiated with different backbones. Ablations confirm that retrieval, posterior checking, and schema constraints make complementary contributions. The parsed subtasks further support demonstration collection and VLA model training, enabling successful execution in both simulation and real-robot settings. ProtoAct thus provides a practical interface between biological protocol understanding and embodied robotic execution.

1 Introduction

Biological wet-lab experiments are commonly guided by protocols, namely written instructions describing how samples are prepared, processed, monitored, and measured. As automated laboratories increasingly use robotic systems to improve experimental efficiency and reproducibility, these protocols provide a natural source of task specifications. However, they are written for trained researchers and are not directly machine-executable: routine operations may be omitted, equivalent actions may be described differently, and parameters often depend on context. Converting protocols into structured, robot-readable action representations is therefore

essential for connecting experimental knowledge with automated execution.

Recent studies have applied large language models to protocol understanding, generation, and planning. BioPlanner converts biological protocols into pseudocode and evaluates the planning ability of large language models using optional pseudo-functions (O’Donoghue et al. 2023). ProtoMed-LLM adopts a predefined set of experimental actions for pseudocode extraction and model evaluation (Yi, Lim, and Yoon 2025). BioMARS integrates protocol generation, pseudocode conversion, and multimodal anomaly detection within a multiagent robotic system (Qiu et al. 2025). These studies show that language models can organize experimental instructions into structured forms. However, pseudocode and free-form plans do not fully specify what embodied execution requires: which physical action to perform, when to trigger it, which parameters to use, and how each step depends on the experimental state.

This gap is especially important for vision-language-action models, which learn robot behavior from visual observations, language instructions, and demonstrations. Equivalent operations may be described differently across protocols, while one sentence may combine several physical actions. Reliable execution thus requires an intermediate representation that preserves protocol meaning while making monitoring conditions, action primitives, parameters, and procedural sequences explicit. It should remain readable for human inspection and also support automatic evaluation, demonstration collection, and policy learning.

To address this need, we introduce ProtoAct, a structured protocol-grounding framework that translates biological procedures into action programs designed for embodied execution. ProtoAct generates monitoring conditions and subtask sequences, checks them against experiment-specific requirements, and grounds the refined procedure in a parameterized action schema. We further construct BioP2E, a manually annotated cell-culture protocol dataset aligned with these embodied task representations. Finally, we validate ProtoAct outputs through robotic task construction, demonstration col-

*These authors contributed equally.

†Corresponding author.

lection, and VLA training. Experiments in simulation and on a physical robot show their utility for embodied data generation and policy execution. The code is available at <https://github.com/gjm112233/ProtoAct>. Our contributions are summarized as follows.

- We introduce, to our knowledge, the first protocol-grounding formulation for embodied wet-lab execution, connecting free-form experimental instructions with state-aware task representations and parameterized robotic actions.
- We construct BioP2E by manually annotating 22 cell-culture protocols into 258 monitoring conditions, 910 executable subtasks, and 962 grounded action calls, providing a resource for evaluation and embodied task construction.
- We establish a multi-level benchmark for biological protocol grounding and further validate the grounded representations through VLA training and execution in both simulation and real-robot settings.

2 Related Work

2.1 Biological Protocol Parsing and Laboratory Automation

Biological experimental protocols typically describe experimental procedures in natural language. Recent studies have explored the use of large language models and agent-based systems for protocol understanding, generation, and structured transformation. BioPlanner formulates protocol generation as pseudo-function selection and stepwise pseudocode reconstruction (O’Donoghue et al. 2023). ProtoMed-LLM evaluates the protocol generation capabilities of large language models using a predefined set of experimental actions (Yi, Lim, and Yoon 2025). ProtoCode converts PCR procedures into machine-readable representations (Jiang et al. 2024). For experimental execution, RoboCulture combines robotic manipulation, visual feedback, and behavior trees to automate long-duration biological culture workflows (Angers et al. 2025). BioProVLA-Agent integrates protocol parsing, visual state verification, and VLA models to support closed-loop execution in wet-lab environments (Du et al. 2026). However, existing studies still lack an intermediate representation that connects protocol parsing with robotic execution. To address this gap, ProtoAct employs a two-stage parsing pipeline that first produces task decompositions for human interpretation and then converts them into structured action representations for robotic execution.

2.2 Action Space Modeling for Robot Execution

Mapping natural language instructions to executable action sequences is a key step in connecting large language models with robotic systems. SayCan combines language-model-based semantic scores with robotic skill affordances to select skills that are both semantically appropriate and executable (Ichter et al. 2023). Code as Policies generates programs that invoke control APIs, whereas ProgPrompt explicitly provides the available action set to guide task planning under environmental constraints (Liang et al. 2023; Singh et al.

2023). ProtoMed-LLM uses a predefined set of experimental actions to standardize the generation and evaluation of protocol pseudo-functions (Yi, Lim, and Yoon 2025). However, existing methods provide limited modeling of state dependencies and procedural constraints in biological wet-lab experiments. To address this limitation, ProtoAct uses monitoring conditions and subtask sequences as intermediate representations. ActSchema further defines monitoring primitives, action primitives, and their parameter schemas, enabling these representations to be mapped into JSON action function sequences for automated evaluation and robotic task construction.

2.3 Robotic Manipulation Learning and Simulation Verification Based on VLA Models

Vision-language-action models jointly process visual observations, language instructions, and robot states to generate language-conditioned robot actions. RT-2 improves generalization by jointly fine-tuning a pretrained vision-language model on robotic trajectory data (Zitkovich et al. 2023). OpenVLA provides a general-purpose VLA model and fine-tuning framework trained on large-scale robotic demonstrations (Kim et al. 2025). π_0 adopts a flow-matching architecture to learn general manipulation policies (Black et al. 2025), while SmolVLA focuses on lightweight and cost-effective deployment (Shukor et al. 2025). In addition, LIBERO provides a simulation benchmark for lifelong robot learning and knowledge transfer (Liu et al. 2023a), and AutoBio develops a digital simulation platform for biological laboratory automation (Lan et al. 2026). Existing studies have paid limited attention to connecting protocol parsing results with robot learning tasks. This work uses VLA models to validate that the subtasks generated by ProtoAct can support data collection, model training, and execution evaluation in both simulated and real-world robotic tasks, thereby establishing a complete workflow from protocol parsing to embodied execution validation.

3 ProtoAct Pipeline

3.1 Overview

ProtoAct is designed to bridge biological protocol understanding and embodied robotic execution by converting natural-language procedures into structured and verifiable workflow representations at both the subtask and action-function levels, as shown in Figure 1. Given a raw biological experimental protocol as input, the system first combines a large language model with ProtoRAG to parse the protocol into natural-language monitoring conditions and subtask sequences. The monitoring conditions specify the requirements that must be satisfied before subsequent subtasks can be executed, while each subtask represents the smallest physical action that the robot needs to perform under the corresponding condition. This representation preserves the temporal logic among experimental steps and provides clear boundaries for subsequent action mapping.

After obtaining the initial parsing result, ProtoAct applies RefineChecker as a posterior checking mechanism to verify

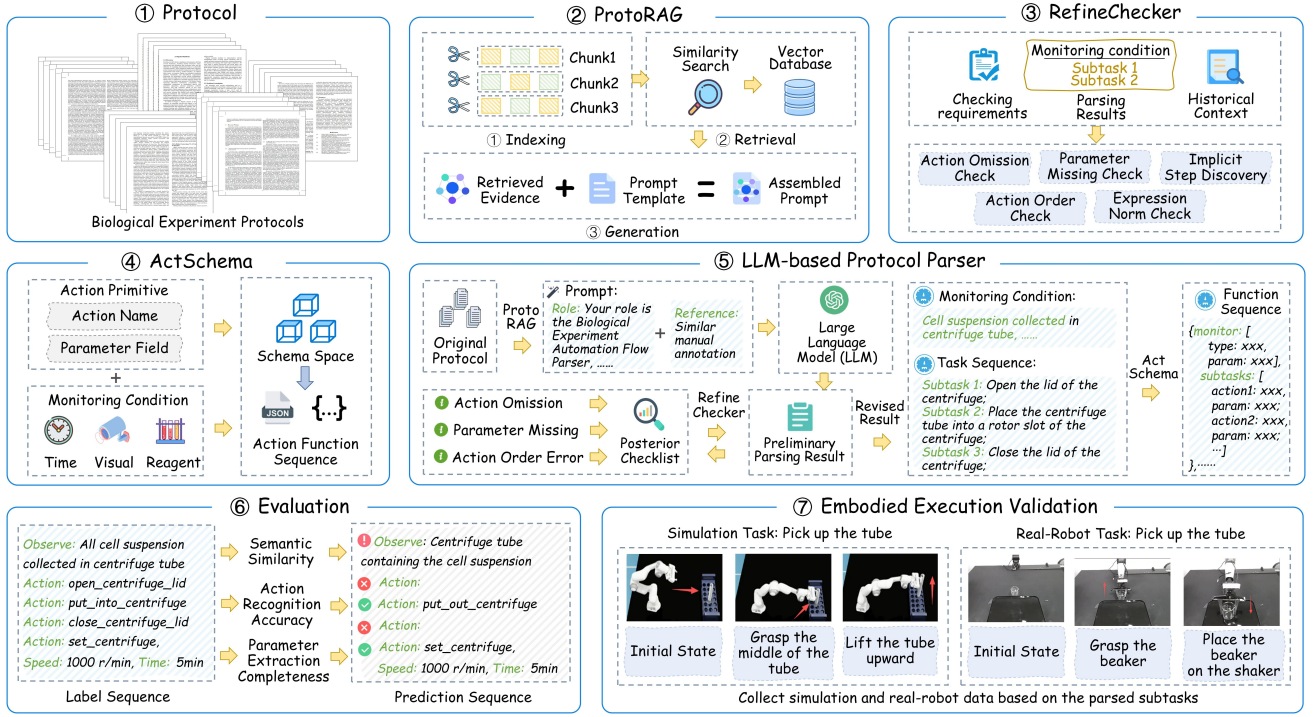


Figure 1: Architecture of the ProtoAct system.

and revise the preliminary result according to experiment-specific notes provided by the user. This process preserves the correctly parsed content while incorporating domain knowledge to address the large language model’s incomplete understanding of experimental logic. Finally, under the constraints of ActSchema, the monitoring conditions and subtask sequences are mapped into structured JSON action function sequences. Compared with directly generating low-level control commands from raw experimental protocols, ProtoAct introduces an intermediate representation between semantic parsing and action function mapping. This design makes the parsing results easier for users to inspect and revise, while also supporting the subsequent conversion into action function sequences with a unified format.

To support knowledge base uploading, protocol input, result display, and multi turn interaction, the system uses Streamlit to build the interactive interface. LangChain is used to organize retrieval, prompt construction, model invocation, and dialogue history management.

3.2 ProtoRAG: Retrieval-Augmented Protocol Parsing

ProtoRAG is used to improve the consistency and completeness of the parsing results. A workflow illustration of ProtoRAG is provided in the supplementary material. ProtoRAG consists of offline knowledge construction and online retrieval-augmented generation. In the offline stage, manually annotated protocol examples are used as the content of the knowledge base. Each example consists of a raw protocol fragment and its corresponding standardized parsing result.

The system first splits the knowledge base text into chunks, then uses an embedding model to encode the text chunks into vectors, which are stored in a local Chroma vector database. In the online parsing stage, the new experimental protocol is used as the query text for the vector retriever. The system computes the semantic similarity between the query text and all text chunks in the knowledge base, and returns the top k document fragments with the highest similarity:

$$R_k(P) = \text{TopK}(\text{sim}(E(P), v_j), k). \quad (1)$$

Here, $R_k(P)$ denotes the top k reference fragments retrieved for the input protocol P , sim denotes the vector similarity function, E denotes the embedding model, v_j denotes the vector representation of the j -th text chunk, and TopK denotes the operation of selecting the k results with the highest similarity.

The retrieved results are then formatted as reference materials and combined with the role definition, action granularity rules, and other instructions to form the final prompt. Let G denote the set of parsing rules. The parsing process of ProtoRAG can be represented as:

$$Y^{(0)} = f_{\text{ProtoRAG}}(P, R_k(P), G). \quad (2)$$

Here, $Y^{(0)}$ denotes the initial monitoring conditions and subtask sequences. P denotes the input raw protocol.

The main role of ProtoRAG is to provide retrievable external priors for protocol parsing. By dynamically retrieving similar cases according to the input, ProtoRAG offers task-relevant references for different experimental procedures and helps improve the quality of the initial parsing result.

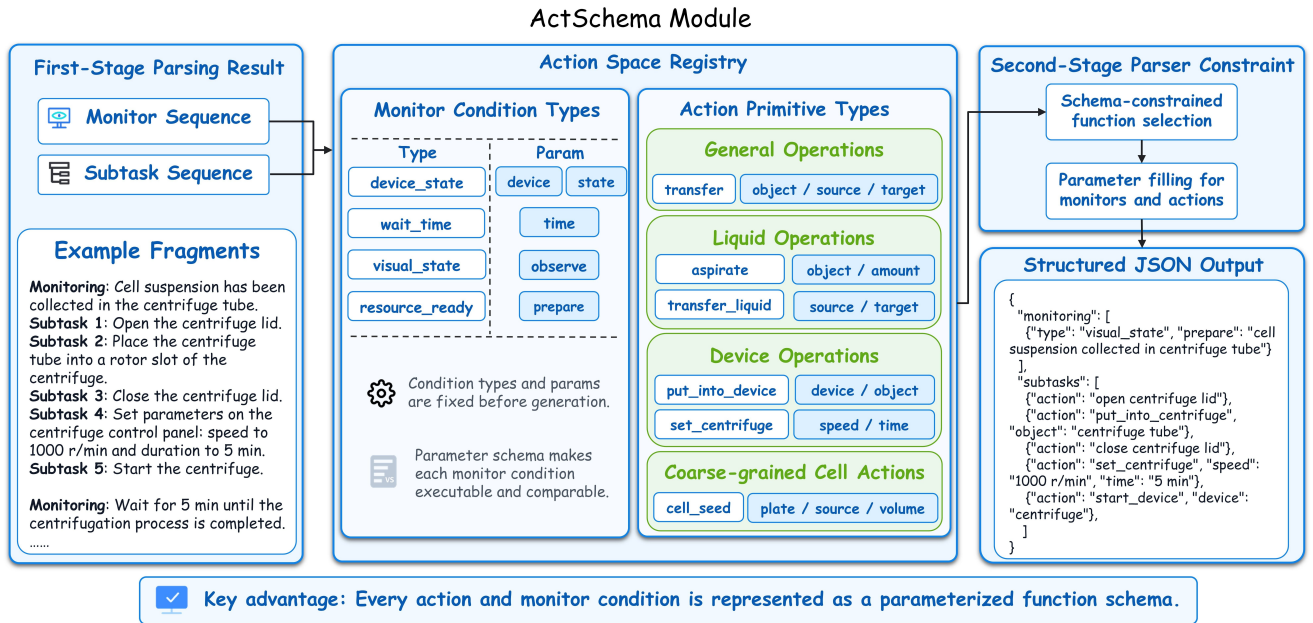


Figure 2: Architecture of the ActSchema module.

3.3 RefineChecker: Post-parsing Verification and Targeted Revision

RefineChecker is used to verify and revise the initial natural language parsing result. An architecture diagram for RefineChecker is included in the supplementary material. The input of RefineChecker consists of two parts: the initial parsing result $Y^{(0)}$, and the checking information U provided by the user for the current experiment. RefineChecker compares $Y^{(0)}$ with U . If the relevant notes have already been covered, the result is kept unchanged. If omissions or errors are identified, the corresponding fragments are reanalyzed, and revised monitoring conditions and subtask sequences are generated.

RefineChecker focuses on four types of parsing defects, as shown in Figure 3. The first type is missing actions, where implicit operations in the raw protocol are absent from the subtask sequences. The second type is missing parameters, where parameter information in the raw protocol is not extracted. The third type is incorrect order, where the parsed result violates the dependency between experimental operations. The fourth type is improper parsing granularity, such as merging multiple consecutive physical actions into a single subtask. The verification process of RefineChecker can be formalized as:

$$Y^{(1)} = f_{\text{RefineChecker}}(Y^{(0)}, U). \quad (3)$$

Here, $Y^{(1)}$ denotes the intermediate representation after post-parsing verification and revision.

To support continuous comparison among the initial parsing result, user provided information, and revised output, ProtoAct maintains dialogue history in the LangChain workflow, allowing the model to access the initial parsing result.

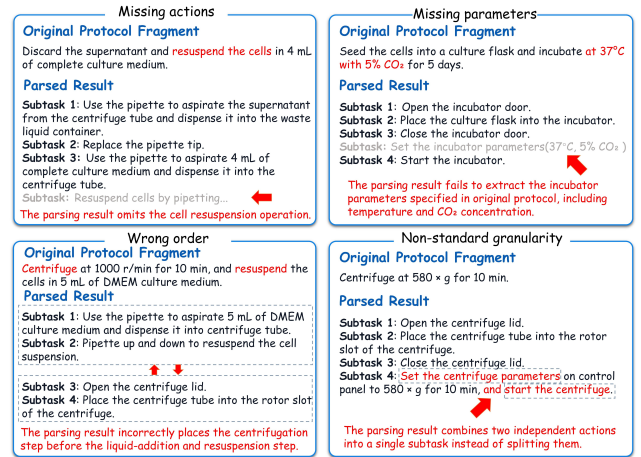


Figure 3: Cases of four types of parsing defects.

This mechanism is lightweight and interactive, making it suitable for forming a feedback loop between human review and model based parsing.

3.4 ActSchema: Constrained Function Selection and Parameter Filling

ActSchema is used to convert the revised intermediate representation $Y^{(1)}$ into a structured action function sequence. Its structure is shown in Figure 2. Although $Y^{(1)}$ is readable and suitable for human inspection, it is still expressed in natural language, making it difficult to use directly for automatic evaluation. To convert these natural language fragments into function calls recognizable by the system, we constructs ActSchema to constrain the process of function

selection and parameter filling.

ActSchema defines monitoring condition types, action primitive names, and the required parameter fields. Based on these schemas, ActSchema maps the natural language monitoring conditions and subtask sequences into structured function calls. Let z_t denote the t -th text unit extracted from $Y^{(1)}$, which contains the monitoring condition m_t and several subtasks $s_{t,j}$. The mapping process of ActSchema can be represented as:

$$a_t = f_{\text{ActSchema}}(z_t, \Omega). \quad (4)$$

Here, a_t denotes the structured function call mapped from the text unit z_t , and Ω denotes the complete function schema space defined by ActSchema.

All function calls are then organized according to the execution order of monitoring conditions and subtasks to form the action function sequence:

$$A = \{a_t\}_{t=1}^N. \quad (5)$$

With ActSchema constraints, ProtoAct formulates open-ended language generation as constrained function selection and parameter filling, facilitating output alignment and automatic evaluation.

To ensure that constraint information can be consistently used during parsing, the system organizes the complete monitoring condition and action primitive schemas into external configuration files, which are provided to the model in the parsing workflow. ActSchema serves as the bridge from natural language task descriptions to standardized function representations in the overall pipeline.

4 Experiment

4.1 BioP2E Dataset

To evaluate structured protocol parsing and its applicability to embodied robotic tasks, we manually annotate 22 public wet-lab cell culture protocols collected from Protocols.io (Teytelman et al. 2016) to construct BioP2E. Each protocol is annotated at two levels: natural-language monitoring conditions and subtask sequences at the semantic level, and schema-constrained JSON action function sequences at the execution level. The detailed annotation procedure is provided in the supplementary material. All annotations are manually reviewed by experts with biological experiment backgrounds for experimental validity, semantic completeness, and executability. Among the 22 annotated protocols, two are reserved as reference protocols for constructing the ProtoRAG knowledge base. The remaining 20 protocols are used exclusively for model evaluation, ensuring that the retrieval knowledge base does not overlap with the evaluation set. Dataset statistics are summarized in Table 1.

4.2 Evaluation Metrics

We evaluate biological protocol parsing by comparing model outputs with manually annotated ground truth at three levels: action primitive and monitoring condition recognition, parameter extraction and value similarity, and action sequence order consistency. A GPT-based evaluator is further used to

Statistic	Value
Biological experiment protocols	22
Raw protocol steps	103
Monitoring primitive types	4
Action primitives	30
Monitoring conditions	258
Subtasks	910
Action functions	962
Average subtasks per protocol	41.36
Average action calls per protocol	43.73

Table 1: Overall statistics of the BioP2E dataset.

assess the overall semantic validity and executability of the parsing results.

Action Primitives and Monitoring Condition Recognition Evaluation For each protocol, the predicted action function sequence is matched against the manually annotated ground-truth sequence. We count true positives (TP), false positives (FP), and false negatives (FN), and report the F1 score for both action primitives and monitoring condition types.

Parameter Field Extraction and Parameter Value Similarity Evaluation Parameter fields are evaluated using F1 under the same matching procedure. Parameter values are evaluated using BLEU (Papineni et al. 2002) and SciBERTScore (Beltagy, Lo, and Cohan 2019), which measure lexical overlap and semantic similarity, respectively.

BLEU measures the n -gram overlap between predicted and ground-truth parameter values:

$$\text{BLEU} = \exp \left(\sum_{n=1}^N w_n \log p_n \right). \quad (6)$$

Here, p_n denotes the matching proportion of continuous token fragments of length n between the two values, w_n denotes the weight assigned to each n -gram order, and N denotes the maximum n -gram order.

Semantically equivalent parameter values may have different surface forms, we further compute their embedding similarity using SciBERT:

$$\text{SciBERTScore} = \frac{1}{N} \sum_{i=1}^N \frac{\langle E(a_i^{\text{pred}}), E(a_i^{\text{GT}}) \rangle}{\|E(a_i^{\text{pred}})\| \|E(a_i^{\text{GT}})\|}. \quad (7)$$

Here, E denotes the SciBERT encoder, a_i^{pred} and a_i^{GT} denote the predicted and ground-truth parameter values, and N denotes the number of evaluated parameters.

Action Sequence Order Consistency Evaluation We use the normalized Levenshtein distance L_{dn} to evaluate the order consistency between predicted and ground-truth action sequences (Levenshtein 1966). Given the predicted action sequence S_{pred} and the ground-truth sequence S_{GT} , it is defined as:

$$L_{\text{dn}} = \frac{L_d(S_{\text{pred}}, S_{\text{GT}})}{|S_{\text{GT}}|}. \quad (8)$$

Here, L_d denotes the minimum number of insertions, deletions, and substitutions required to transform S_{pred} into S_{GT} , and $|S_{\text{GT}}|$ denotes the length of the ground truth action sequence. A lower L_{dn} indicates better sequence consistency.

Parsing Quality Evaluation with a GPT Evaluator Following prior LLM-based evaluation studies (Liu et al. 2023b), we use GPT-5.3 to complement the automatic metrics. The evaluator receives the original protocol, the generated monitoring conditions and subtask sequences, and the predefined parsing rules, to assess the overall semantic validity and executability of the parsing result. An example GPT-evaluator output is included in the supplementary material. This evaluation helps identify experimental validity and potential execution risks that may not be captured by automatic metrics.

4.3 Multi-Model Comparison on Protocol Parsing

To compare the performance of different large language models on biological protocol parsing, we evaluate seven models on 20 cell culture protocols. The evaluation is conducted from two aspects: monitor condition and action primitive.

As shown in Table 2, the F1 scores of different models are generally close in monitor condition type recognition. Among them, deepseek-r1, glm-4.5, and deepseek-v3.2 perform relatively better, indicating that they can more effectively identify monitor conditions from the original protocols and assign them to the correct types. For parameter parsing, deepseek-v3.2 performs better in monitor condition parameter field extraction, while gpt-5.3 achieves better similarity for monitor parameter values. For action parsing, gpt-5.3 achieves the best results in action primitive F1, action parameter F1, and L_{dn} , suggesting that its outputs are closer to the manual annotations in terms of action coverage, parameter field extraction, and order consistency. Qwen3-max performs best in action parameter value similarity, indicating that its generated parameter values are closer to the manual annotations in both lexical form and semantic meaning.

4.4 Ablation Experiments

To analyze the contribution of each module in the ProtoAct pipeline, we conduct ablation experiments on ProtoRAG, RefineChecker, and ActSchema using Qwen3-Max as the base model. Qwen3-Max is selected for its strong overall performance in the multi-model comparison. The complete ablation results are reported in the supplementary material.

The full model achieves the best results in monitor condition parsing, action primitive recognition, parameter extraction, and sequence consistency, indicating that the three modules all contribute to the overall parsing performance. Specifically, removing ProtoRAG leads to the most obvious decrease in monitor condition-related metrics, suggesting that reference examples help the model identify implicit monitor conditions and their parameters that depend on experimental context. Removing ActSchema causes the largest drop in action primitive F1 and action parameter F1, showing

that the predefined action schema space can constrain action function generation and improve the regularity of parameter structures. Removing RefineChecker increases omissions in monitor conditions and action primitives and raises L_{dn} , indicating that this module can supplement missing content through posterior checking and help maintain a reasonable experimental procedure order.

4.5 Comprehensive Comparison of Parsing Quality Based on the GPT Evaluator

We use GPT-5.3 to evaluate the monitoring conditions and subtask sequences generated by different models in the first parsing stage and to compare the effects of ProtoRAG and RefineChecker on parsing quality.

The GPT evaluation results reported in the supplementary material show that introducing RefineChecker improves the scores of most models, indicating that posterior checking helps correct action omissions and procedural ordering errors in the parsing results. In the vascular endothelial cell protocol, Qwen3-Max with RefineChecker recovers the implicit transfer operation before centrifugation but misses the removal of the washing solution, resulting in a score of 4. With ProtoRAG, the model uses annotated examples to complete the washing procedure and achieves a score of 5. Overall, ProtoRAG and RefineChecker provide complementary improvements in parsing completeness and consistency.

4.6 Simulation and Real-Robot Validation for Embodied Execution

Simulation Dataset To validate whether the subtasks parsed by ProtoAct can be used for embodied execution, we construct five robotic manipulation tasks in Pipette, an embodied simulation platform for wet-lab robotics (Liu et al. 2026). The left side of Figure 4 shows their key execution stages. For each task, 30 trajectories are collected through keyboard teleoperation and augmented by varying scene illumination and execution speed. Additional dataset statistics are listed in the supplementary material.

Simulation Execution Results Using the five simulation task datasets, we train SmolVLA and π_0 with the LeRobot framework (Cadene et al. 2026) and evaluate them online in Pipette (Liu et al. 2026). The supplementary material details the evaluation workflow, and the results are reported in Table 3. Overall, SmolVLA achieves higher execution success rates on most tasks, while π_0 also completes successful executions. The results show that the subtasks parsed by ProtoAct from biological experimental protocols can guide users without biological expertise in collecting robotic demonstration data and can also serve as language-conditioned inputs for VLA model training and execution, thereby validating their applicability to embodied manipulation tasks.

Real-Robot Execution Validation To validate the subtasks generated by ProtoAct on a physical robot arm, we construct three real-robot tasks. The right side of Figure 4 shows their key execution stages. Together, they form a continuous workflow. For each task, 100 demonstration trajectories are collected through gamepad teleoperation and used to

Models	Monitor Type	Monitor Param.			Action Prim.	Action Param.			Action Order
	F1	F1	BLEU	SciBERT	F1	F1	BLEU	SciBERT	L_{dn}
qwen3-max	0.88±0.06	0.89±0.05	0.74±0.12	0.95±0.03	0.93±0.06	0.94±0.05	0.87±0.07	0.98±0.01	0.13±0.10
qwen-max	0.88±0.11	0.89±0.10	0.73±0.11	0.93±0.03	0.92±0.05	0.93±0.04	0.81±0.11	0.94±0.02	0.17±0.10
gpt-5.3	0.89±0.06	0.90±0.06	0.77±0.09	0.96±0.02	0.95±0.04	0.96±0.03	0.85±0.07	0.97±0.01	0.09±0.10
deepseek-v3.2	0.90±0.07	0.92±0.06	0.71±0.11	0.94±0.02	0.93±0.05	0.95±0.04	0.84±0.09	0.95±0.02	0.14±0.11
deepseek-r1	0.90±0.07	0.91±0.07	0.71±0.10	0.94±0.02	<u>0.94±0.05</u>	0.94±0.05	0.84±0.07	0.94±0.02	0.13±0.10
kimi-k2	0.88±0.08	0.89±0.08	0.72±0.13	0.92±0.03	0.92±0.06	0.91±0.05	0.82±0.11	0.93±0.01	0.11±0.10
glm-4.5	0.90±0.08	0.90±0.07	0.70±0.11	0.94±0.02	<u>0.94±0.05</u>	0.93±0.04	0.84±0.07	0.96±0.01	<u>0.11±0.09</u>

Table 2: Comparison of protocol parsing performance across monitor conditions and action primitives. Each cell reports the mean and standard deviation of the corresponding metric. The best and second-best results are marked in bold and underlined.

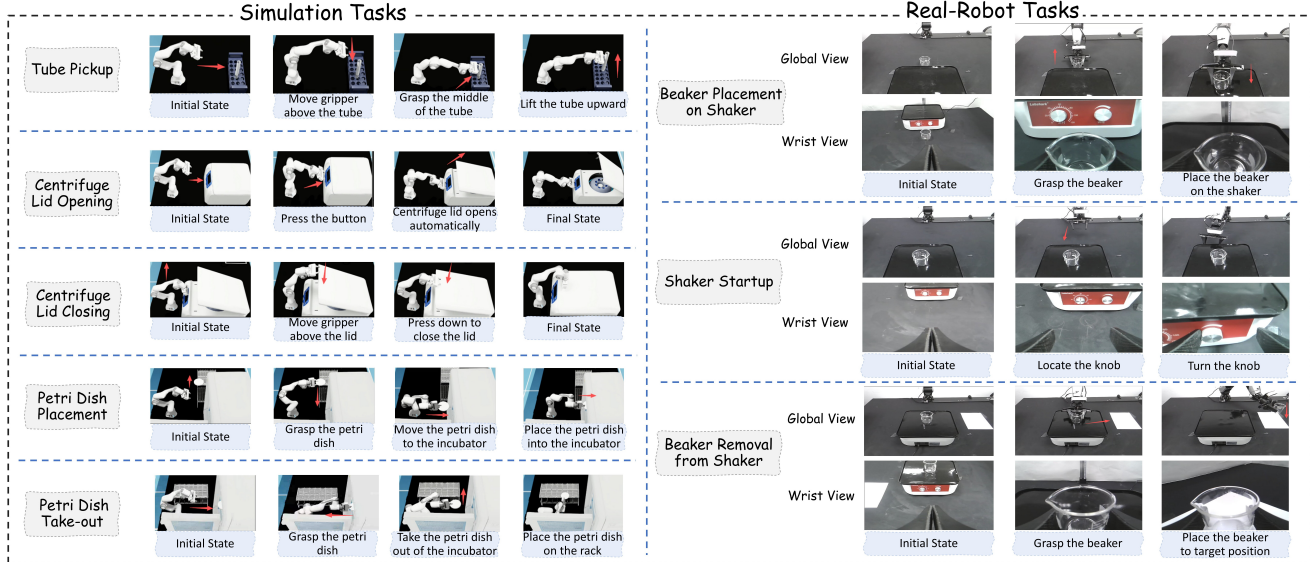


Figure 4: Example embodied execution tasks in simulation and real-robot settings.

Task	Episodes	π_0	SmolVLA
Tube pickup	100	59	95
Centrifuge lid opening	100	90	96
Centrifuge lid closing	100	96	98
Petri dish take-out	100	34	66
Petri dish placement	100	52	96

Table 3: Execution success rates of two VLA models on five simulation tasks.

train SmolVLA and π_0 with the LeRobot framework. Both models complete the tasks, with SmolVLA showing more stable execution, consistent with the simulation results. The experiments show that the subtasks generated by ProtoAct can be converted into executable robotic actions in physical environments, validating the practical applicability of the proposed pipeline in connecting protocol understanding with real-world embodied execution.

5 Conclusion

This paper introduces ProtoAct, a structured protocol-grounding framework that bridges free-form biological experimental protocols and embodied robotic execution. ProtoAct first produces a human-inspectable representation consisting of monitoring conditions and fine-grained subtasks, and then grounds it into schema-constrained JSON action functions. Through ProtoRAG, RefineChecker, and ActSchema, the framework makes experimental states, action boundaries, parameter details, and procedural dependencies explicit, while supporting both automatic evaluation and downstream robotic execution. We further construct BioP2E by manually annotating 22 cell-culture protocols, and establish a multi-level benchmark covering action recognition, parameter extraction, and sequence consistency. Experiments across seven large language models demonstrate that ProtoAct can be instantiated with different model backbones. The ablation results confirm the complementary contributions between modules. Simulation and real-robot experiments further show that the generated subtasks can support demonstration collection, VLA model training, and physical

robot execution, providing a practical connection between biological protocol understanding and embodied action.

BioP2E is currently limited in scale and biological coverage, RefineChecker depends on experiment-specific information provided by users, and the robotic evaluation focuses on selected manipulation tasks rather than complete closed-loop wet-lab workflows. Future work will expand the protocol domains and action schema, incorporate structured experimental knowledge and autonomous error-recovery mechanisms, and evaluate longer workflows in real laboratory environments.

References

- Angers, K.; Darvish, K.; Yoshikawa, N.; Okhovatian, S.; Bannerman, D.; Yakavets, I.; Shkurti, F.; Aspuru-Guzik, A.; and Radisic, M. 2025. RoboCulture: A Robotics Platform for Automated Biological Experimentation. arXiv:2505.14941.
- Beltagy, I.; Lo, K.; and Cohan, A. 2019. SciBERT: A Pre-trained Language Model for Scientific Text. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, 3615–3620.
- Black, K.; Brown, N.; Driess, D.; Esmail, A.; Equi, M. R.; Finn, C.; Fusai, N.; Groom, L.; Hausman, K.; Ichter, B.; Jakubczak, S.; Jones, T.; Ke, L.; Levine, S.; Li-Bell, A.; Mothukuri, M.; Nair, S.; Pertsch, K.; Shi, L. X.; Smith, L.; Tanner, J.; Vuong, Q.; Walling, A.; Wang, H.; and Zhilinsky, U. 2025. π_0 : A Vision-Language-Action Flow Model for General Robot Control. In *Proceedings of Robotics: Science and Systems*.
- Cadene, R.; Alibert, S.; Capuano, F.; Aractingi, M.; Zouitine, A.; Kooijmans, P.; Choghari, J.; Russi, M.; Pascal, C.; Palma, S.; Aubakirova, D.; Shukor, M.; Moss, J.; Soare, A.; Lhoest, Q.; Gallouédec, Q.; and Wolf, T. 2026. LeRobot: An Open-Source Library for End-to-End Robot Learning. In *The Fourteenth International Conference on Learning Representations*.
- Du, Z.; Wang, Z.; Fei, H.; Cao, X.; Xiao, T.; Wang, Q.; Jin, H.; Gu, J.; Lu, Q.; and Liu, Z. 2026. BioProVLA-Agent: An Affordable, Protocol-Driven, Vision-Enhanced VLA-Enabled Embodied Multi-Agent System with Closed-Loop-Capable Reasoning for Biological Laboratory Manipulation. arXiv:2605.07306.
- Ichter, B.; Brohan, A.; Chebotar, Y.; Finn, C.; Hausman, K.; Herzog, A.; Ho, D.; Ibarz, J.; Irpan, A.; Jang, E.; Julian, R.; Kalashnikov, D.; Levine, S.; Lu, Y.; Parada, C.; Rao, K.; Sermanet, P.; Toshev, A. T.; Vanhoucke, V.; Xia, F.; Xiao, T.; Xu, P.; Yan, M.; Brown, N.; Ahn, M.; Cortes, O.; Sievers, N.; Tan, C.; Xu, S.; Reyes, D.; Rettinghouse, J.; Quiambao, J.; Pastor, P.; Luu, L.; Lee, K.-H.; Kuang, Y.; Jesmonth, S.; Joshi, N. J.; Jeffrey, K.; Ruano, R. J.; Hsu, J.; Gopalakrishnan, K.; David, B.; Zeng, A.; and Fu, C. K. 2023. Do as I Can, Not as I Say: Grounding Language in Robotic Affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205, 287–318.
- Jiang, S.; Evans-Yamamoto, D.; Bersenev, D.; Palaniappan, S. K.; and Yachie, A. 2024. ProtoCode: Leveraging Large Language Models (LLMs) for Automated Generation of Machine-Readable PCR Protocols from Scientific Publications. *SLAS Technology*, 29: 100134.
- Kim, M. J.; Pertsch, K.; Karamcheti, S.; Xiao, T.; Balakrishna, A.; Nair, S.; Rafailov, R.; Foster, E. P.; Sanketi, P. R.; Vuong, Q.; Kollar, T.; Burchfiel, B.; Tedrake, R.; Sadigh, D.; Levine, S.; Liang, P.; and Finn, C. 2025. OpenVLA: An Open-Source Vision-Language-Action Model. In *Proceedings of The 8th Conference on Robot Learning*, volume 270, 2679–2713.
- Lan, Z.; Jiang, Y.; Wang, R.; Xie, X.; Zhang, R.; Zhu, Y.; Li, P.; Yang, T.; Chen, T.; Gao, H.; Yang, X.; Li, X.; Zhang, H.; Mu, Y.; and Luo, P. 2026. AutoBio: A Simulation and Benchmark for Robotic Automation in Digital Biology Laboratory. In *The Fourteenth International Conference on Learning Representations*.
- Levenshtein, V. I. 1966. Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. *Soviet Physics Doklady*, 10(8): 707–710.
- Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2023. Code as Policies: Language Model Programs for Embodied Control. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 9493–9500.
- Liu, B.; Zhu, Y.; Gao, C.; Feng, Y.; Liu, Q.; Zhu, Y.; and Stone, P. 2023a. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. In *Advances in Neural Information Processing Systems*, volume 36.
- Liu, Y.; Iter, D.; Xu, Y.; Wang, S.; Xu, R.; and Zhu, C. 2023b. G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2511–2522.
- Liu, Z.; Jin, H.; Du, Z.; Wang, Z.; Zhou, D.; Pan, M.; Xu, H.; Li, P.; Gu, J.; Lu, Q.; Wang, Q.; Ji, B.; and Xiao, T. 2026. Pipette: An Embodied Simulation Platform, Benchmark, and Data-Efficient Augmentation Framework for Wet-Lab Robotics. arXiv:2606.12936.
- O’Donoghue, O.; Shtedritski, A.; Ginger, J.; Abboud, R.; Ghareeb, A.; and Rodrigues, S. G. 2023. BioPlanner: Automatic Evaluation of LLMs on Protocol Planning in Biology. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2676–2694.
- Papineni, K.; Roukos, S.; Ward, T.; and Zhu, W.-J. 2002. BLEU: A Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 311–318.
- Qiu, Y.; Huang, Z.; Wang, Z.; Liu, H.; Qiao, Y.; Hu, Y.; Sun, S.; Peng, H.; Xu, R. X.; and Sun, M. 2025. BioMARS: A Multi-Agent Robotic System for Autonomous Biological Experiments. arXiv:2507.01485.
- Shukor, M.; Aubakirova, D.; Capuano, F.; Kooijmans, P.; Palma, S.; Zouitine, A.; Aractingi, M.; Pascal, C.; Russi, M.; Marafioti, A.; Alibert, S.; Cord, M.; Wolf, T.; and Cadene, R. 2025. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics. arXiv:2506.01844.

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2023. Prog-Prompt: Generating Situated Robot Task Plans using Large Language Models. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 11523–11530.

Teytelman, L.; Stoliartchouk, A.; Kindler, L.; and Hurwitz, B. L. 2016. protocols.io: Virtual Communities for Protocol Development and Discussion. *PLOS Biology*, 14(8): e1002538.

Yi, S.; Lim, J.; and Yoon, J. 2025. ProtoMed-LLM: An Automatic Evaluation Framework for Large Language Models in Medical Protocol Formulation. arXiv:2410.04601.

Zitkovich, B.; Yu, T.; Xu, S.; Xu, P.; Xiao, T.; Xia, F.; Wu, J.; Wohllhart, P.; Welker, S.; Wahid, A.; Vuong, Q.; Vanhoucke, V.; Tran, H.; Soricut, R.; Singh, A.; Singh, J.; Sermanet, P.; Sanketi, P. R.; Salazar, G.; Ryoo, M. S.; Reymann, K.; Rao, K.; Pertsch, K.; Mordatch, I.; Michalewski, H.; Lu, Y.; Levine, S.; Lee, L.; Lee, T.-W. E.; Leal, I.; Kuang, Y.; Kalashnikov, D.; Julian, R.; Joshi, N. J.; Irpan, A.; Ichter, B.; Hsu, J.; Herzog, A.; Hausman, K.; Gopalakrishnan, K.; Fu, C.; Florence, P.; Finn, C.; Dubey, K. A.; Driess, D.; Ding, T.; Choromanski, K. M.; Chen, X.; Chebotar, Y.; Carbajal, J.; Brown, N.; Brohan, A.; Arenas, M. G.; and Han, K. 2023. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Proceedings of The 7th Conference on Robot Learning*, volume 229, 2165–2183.

Appendix

A ProtoRAG Workflow

ProtoRAG is used to improve the standardization and completeness of first-stage protocol parsing. Its structure is shown in Figure 5. Biological experimental protocols are highly domain-dependent: the same type of operation may be expressed differently across texts, and without example-based constraints, large language models may produce overly coarse action decompositions or expressions that do not follow the annotation standard. Therefore, this study builds an external knowledge base from manually annotated experimental protocol fragments. When parsing a new protocol, the system retrieves reference fragments that are semantically similar to the current input and provides them to the large language model together with the parsing rules, helping the model generate monitoring conditions and subtask sequences that better match the task requirements.

ProtoRAG consists of two processes: offline knowledge construction and online retrieval-augmented generation. In the offline stage, manually annotated protocol examples are used as the knowledge base content, where each example contains the original experimental steps and the corresponding standardized parsing result. Long texts are split into chunks, and an embedding model encodes these text fragments into vectors, which are stored in a local Chroma vector database. In the online parsing stage, the raw experimental protocol entered by the user first serves as the query text for the vector retriever. The system returns several document fragments with the highest semantic similarity from the vector database. The retrieved results are then formatted as reference materials and assembled with the role definition, action granularity

rules, and other instructions in the system prompt to form the final prompt. When generating the parsing result, the large language model relies not only on its general language understanding ability, but also on the manually annotated results of similar experimental protocols, thereby learning how to organize monitoring conditions, determine subtask granularity, and express different experimental operations in a standardized form.

B RefineChecker Architecture

RefineChecker is used to perform posterior checking and targeted revision on the preliminary natural-language parsing result. Its structure is shown in Figure 6. Because biological experimental protocols often contain abbreviations, omissions, and implicit operations, large language models may still misunderstand certain key steps even with the support of reference examples provided by ProtoRAG. RefineChecker is designed to use user-provided experimental notes to check the generated parsing result item by item.

The input of RefineChecker consists of two parts: the monitoring conditions and subtask sequences initially generated by the model, and the checking information supplemented by the user for the current experiment. This information includes key operation flows in the protocol, experimental constraints, and implicit actions. Based on the user-provided checking information, the model compares it with the historical parsing result. If the original parsing result already covers the relevant notes, no modification is needed. If omissions or errors are confirmed, the erroneous fragment is reanalyzed, and the revised monitoring conditions and subtask fragments are output.

C BioP2E Annotation and Dataset Statistics

Annotation Procedure The annotation process of BioP2E consists of two stages, as shown in Figure 7. In the first stage, the raw protocol is parsed into a natural-language monitoring conditions and subtask sequences. The monitoring sequence describes experimental states or conditions that trigger subsequent operations, such as the completion of device operation, waiting for a specified duration, observing a particular cell state, or confirming that experimental resources are ready. The subtask sequence describes individual actions executed by the robotic arm, such as aspirating liquid, placing a container, putting an object into a centrifuge, setting device parameters, or starting a device. The main goal of this stage is to decompose complex experimental procedures into operation units with clear execution boundaries and experimental semantics, thereby providing a natural-language task representation for downstream robotic understanding and execution.

In the second stage, the monitoring conditions and subtask sequences are further mapped into JSON action function sequences. This stage mainly supports automated evaluation. By converting both manually annotated results and large-language-model-generated parsing results into structured action function sequences, the framework can compute metrics such as action matching, parameter extraction, parameter-value similarity, and action-order consistency. To avoid incomparable outputs caused by freely generated action names,

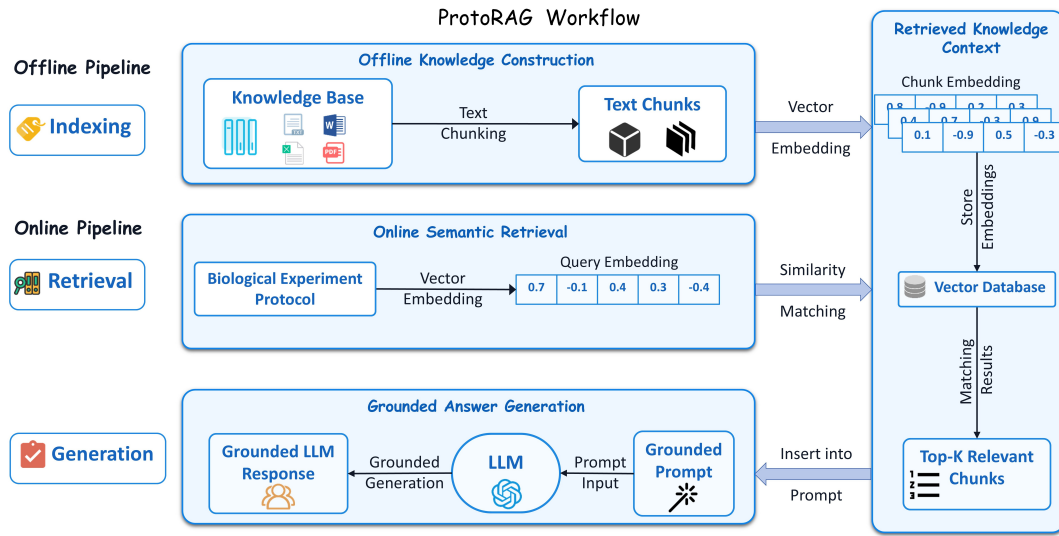


Figure 5: Workflow of the ProtoRAG module.

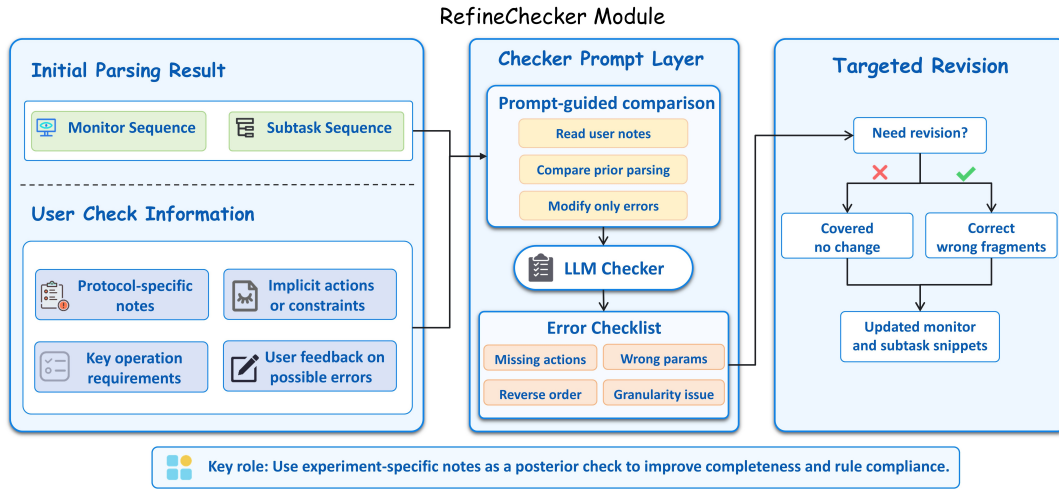


Figure 6: Architecture of the RefineChecker module.

this paper predefines an action space for biological experimental operations. As summarized in Table 4, the action space contains four action types: general manipulation actions, liquid-handling actions, device-operation actions, and coarse-grained biological actions. Each action function has a fixed action name and parameter fields, ensuring consistency in both semantic expression and evaluation computation.

Dataset Scale and Distribution To present the scale and distributional characteristics of BioP2E, we conduct statistical analyses of the dataset from multiple perspectives. Figure 8 shows the distribution of different action primitives used in selected BioP2E protocols. Actions such as *aspirate*, *transfer*, *discard_medium*, *start_device*, and *put_into_incubator* appear frequently in most protocols, indicating that liquid handling, transfer operations, and device operations are typical actions in cell culture exper-

iments. Overall, this heatmap shows that BioP2E contains diverse action primitives and clear inter-protocol variation, providing a fine-grained data basis for evaluating the action-mapping ability of large language models and constructing downstream embodied robotic tasks.

Figure 9 shows the distribution of action types across BioP2E protocols. The figure indicates that liquid-handling and device-operation actions account for relatively large proportions in most protocols, while some protocols also contain a certain proportion of coarse-grained biological actions. This demonstrates that the manual annotations in BioP2E cover not only basic experimental operations but also key procedural stages with biological experimental semantics.

Figure 10 shows the distribution of subtask counts across BioP2E protocols. Simpler protocols are parsed into around 20 subtasks, whereas more complex protocols can be decomposed into more than 70 subtasks. This indicates that BioP2E

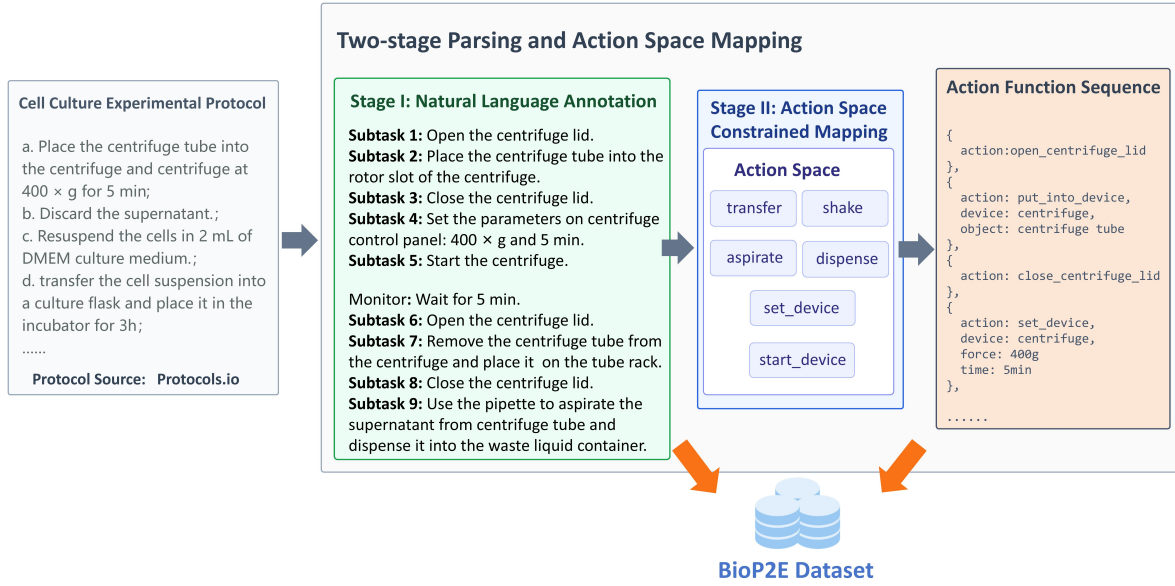


Figure 7: Annotation procedure of the BioP2E dataset.

Category	Example Action	Required Params and Meanings	#Action	#Param.
General manipulation actions	pick_and_place	object: item to move, e.g., culture dish; place: target location, e.g., workbench	4	8
	shake	object: item to shake, e.g., centrifuge tube		
Liquid-handling actions	aspirate	object: liquid to aspirate, e.g., DMEM culture medium; amount: volume, e.g., 6 mL	5	8
	dispense	place: target container or location, e.g., culture dish; amount: volume, e.g., 6 mL		
Device-operation actions	put_into_centrifuge	object: item to load, e.g., centrifuge tube	10	15
	set_device	device: target device, e.g., centrifuge; speed: device setting, e.g., 800 r/min; time: duration, e.g., 10 min		
	take_out_centrifuge	object: item to unload, e.g., centrifuge tube		
Coarse-grained biological actions	resuspension	object: sample or container to resuspend, e.g., centrifuge tube	11	25
	cell_seed	source: source container, e.g., centrifuge tube; place: destination container, e.g., culture flask; volume: operation volume, e.g., 2 mL		

Table 4: Action-space statistics and representative action schemas.

contains many fine-grained and executable experimental operation units.

These statistics show that although BioP2E contains a limited number of raw protocols, each protocol is decomposed into a large number of executable subtasks, resulting in densely structured annotations. Therefore, the core value of BioP2E lies in expanding a small set of professional experimental protocols into multi-level annotated data containing monitoring conditions, subtask sequences, and structured action functions, thereby providing a unified data basis for large-language-model-based protocol parsing, automated evaluation, and embodied robotic task validation.

D GPT Evaluation Case

Figure 11 shows an example output produced by the GPT evaluator. The output consists of four parts. Score denotes the final rating, ranging from 1 to 5, with increments of 0.5 allowed for intermediate levels. Advantages summarizes the experimental stages and key operations correctly covered by the parsing result. Main Problems identifies issues such as rule violations, missing actions and parameters, incorrect procedural order, or inappropriate operation granularity. Summary provides an overall assessment of the parsing quality and the rationale behind the assigned score. The meanings of different scores are as follows: A score of 5 indicates that the parsed result is complete, accurate, and strictly follows

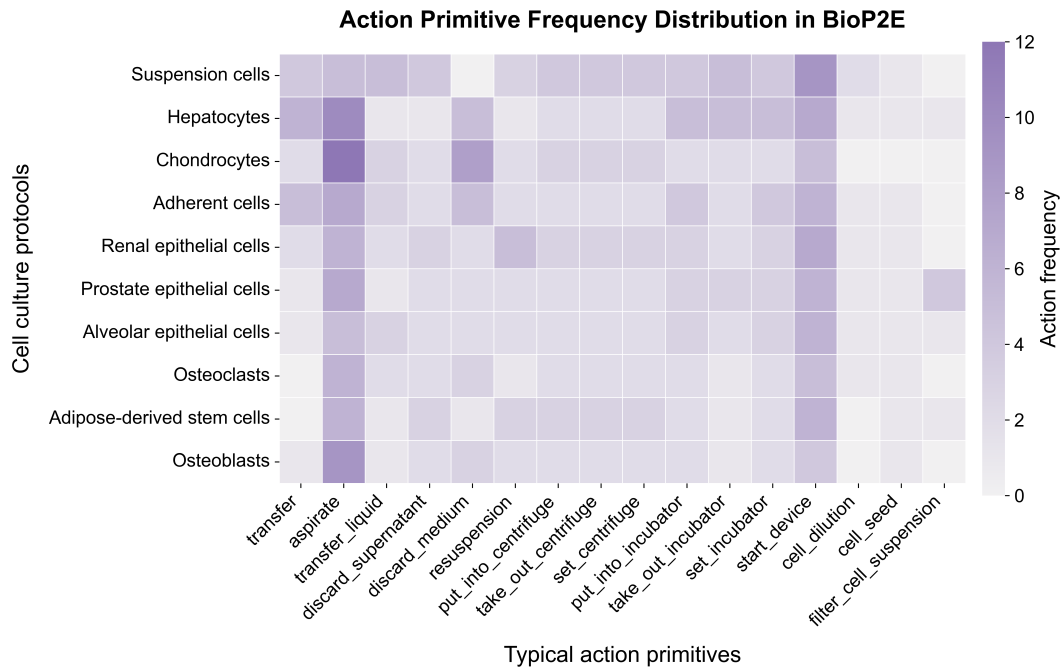


Figure 8: Action primitive frequency heatmap for selected BioP2E cell culture protocols.

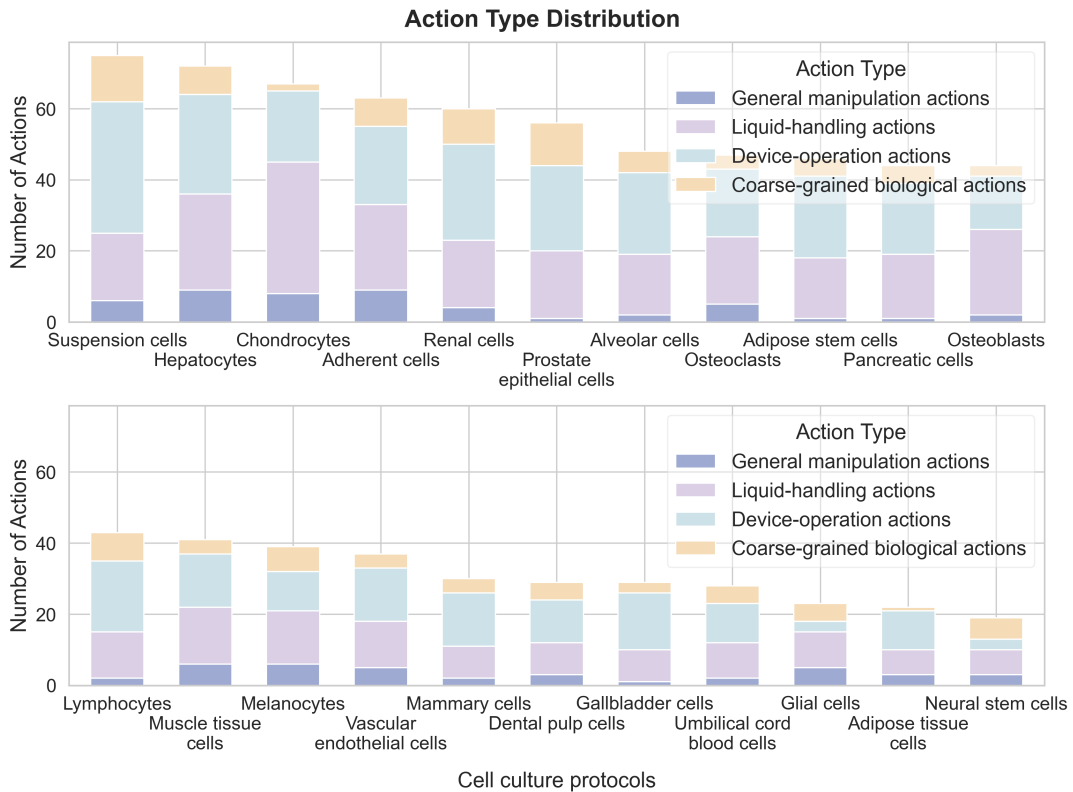


Figure 9: Action type distribution across BioP2E cell culture protocols.

the parsing rules. Scores of 4 or 4.5 indicate that the overall procedure is correct, with only minor omissions or limited

detail level issues. Scores of 3 or 3.5 indicate that the procedure is generally correct but contains several noticeable

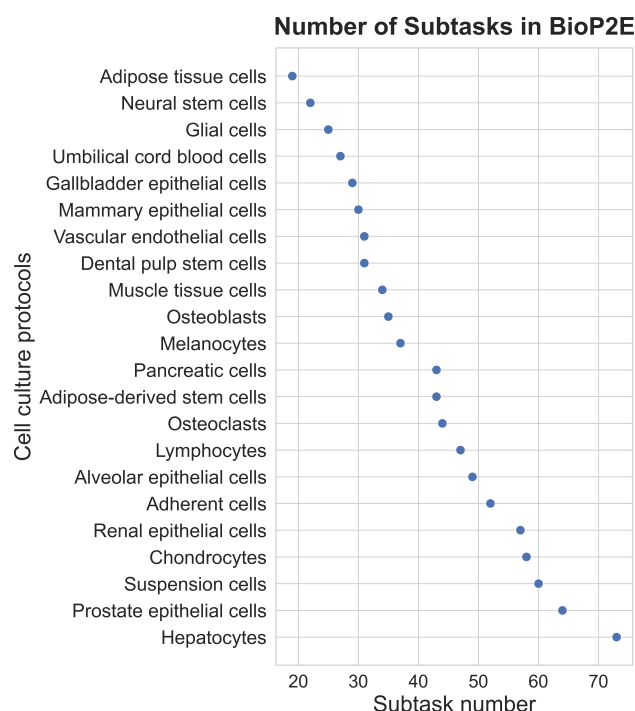


Figure 10: Distribution of manually annotated subtask counts across BioP2E cell culture protocols. Each point denotes one protocol; the x-axis indicates the number of natural-language subtasks, and the y-axis sorts protocols in ascending order by subtask count.

errors or some rule violations. Scores of 2 or 2.5 indicate that the procedure has substantial structural or logical problems, while still retaining partially usable content. Scores of 1 or 1.5 indicate that the parsed result contains severe overall errors, disordered structure, clear rule violations, or omissions of key steps.

E Ablation Results

Tables 6 and 7 provide the complete ablation results for ProtoRAG, ActSchema, and RefineChecker. Since the main text already discusses the key trends, this section only reports the complete tables and briefly summarizes the GPT-based ablation results.

Figure 12 presents the overall scores assigned by the GPT evaluator to the natural-language parsing results generated by different models and module combinations. The evaluated outputs are the monitoring conditions and subtask sequences produced in the first parsing stage. Therefore, this section mainly compares the effects of ProtoRAG and RefineChecker on parsing quality. As shown in the figure, models using only ProtoRAG obtain relatively lower scores. The reason is that the retrieved reference examples cannot fully prevent missing actions, incomplete procedures, or rule violations. After introducing RefineChecker, most models achieve higher scores across different cell protocol parsing tasks, indicating that the posterior checking mechanism helps models identify and revise key omissions and procedural-order errors.

Score: 3 (qwen3-max + RAG → Gallbladder ECs)

Basically correct, but with several obvious issues or rule violations.

Advantages:

- 1: Basically covers the main workflow in the first half, including collagenase digestion, addition of culture medium, centrifugation, and supernatant removal.
- 2: Successfully identifies the cell counting step before cell dilution.
- 3: Reflects the medium change operation on day 3.

Main Problems:

- 1: Subtask 10: **Set the centrifuge** to 1000 r/min for 5 min and **start the centrifuge**.

Main issue: Setting the centrifuge parameters and starting the centrifuge are merged into a single subtask, which violates the rule of stepwise parsing.

- 2: Monitoring condition: The **supernatant in the centrifuge tube is basically removed**.

Subtask 13: Use the pipette to **aspirate 10μL of cell suspension from the centrifuge tube**.

Main issue: After supernatant removal, the model does not explicitly add culture medium, but directly uses the "cell pellet resuspension" for counting, making the source of the counting sample unclear.

- 3: Subtasks 25: Place the culture flask back into the incubator.

Monitoring condition: Wait until the total incubation time reaches 5 days, then end the experiment.

Main issue: The parsed result only states "place the culture flask back into the incubator", but does not reset the incubator parameters and start the incubator. After the additional 2-day incubation, it also lacks the step of "taking the culture flask out of the incubator".

Summary: The parsed result covers the main experimental stages, but it has several obvious rule-related issues, including merged device operations, missing resuspension before cell counting, incomplete incubation steps. Therefore, it is scored 3 points.

Figure 11: Output example of GPT-based parsing quality evaluation.

Overall, the GPT-evaluator results verify the complementary roles of ProtoRAG and RefineChecker in improving protocol parsing quality. ProtoRAG provides manually annotated examples and relevant contextual information, making it easier for models to learn the standard parsing format, action decomposition granularity, and expression style of experimental procedures. RefineChecker further reviews the preliminary parsing results according to predefined rules and experiment-specific notes, thereby reducing missing actions and procedural misalignment. When the two modules are combined, the resulting parses are of higher quality, which is consistent with the conclusions drawn from the ablation experiments.

Model Variant Comparison by Cell Type

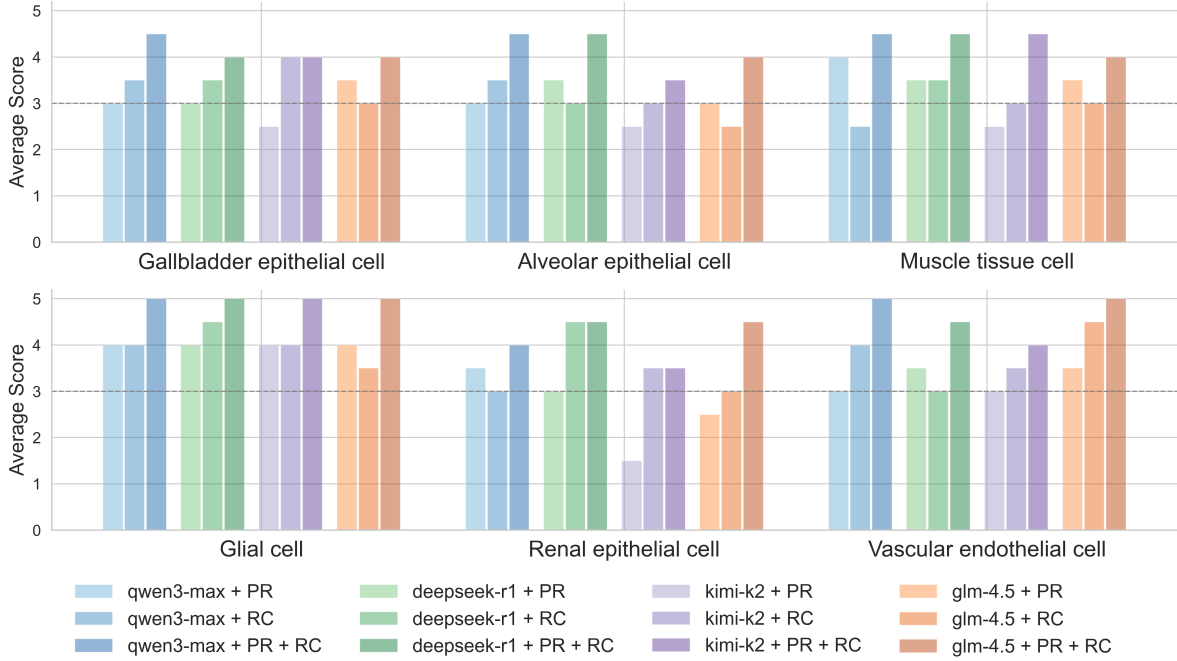


Figure 12: GPT score comparison for model-module combinations across six cell culture protocol parsing tasks. PR and RC denote ProtoRAG and RefineChecker, respectively.

F Simulation Dataset Details

This study constructs five robotic-arm manipulation tasks in the Isaac Sim simulation platform, including picking up the tube from the tube rack, opening the centrifuge lid, closing the centrifuge lid, taking a culture dish out of the incubator, and placing a culture dish into the incubator. These tasks are all derived from typical subtasks in the ProtoAct parsing results.

During data collection, each task is demonstrated by controlling the robotic arm through keyboard teleoperation. The robot joint states, visual observations, and language instructions are saved as HDF5 data. Table 5 shows the dataset statistics for the five simulation tasks. For each task, 30 trajectories are collected through keyboard teleoperation, and the collected data are then augmented offline to improve training-data diversity. The augmentation process mainly expands the data from visual aspects, including adjusting scene lighting intensity and changing trajectory execution speed, thereby improving the adaptability of VLA models to different observation conditions and action rhythms.

It should be noted that the numbers of augmented samples are not exactly the same across tasks because each augmented trajectory is automatically checked according to the success criterion of the corresponding task. If an augmented trajectory fails to satisfy the task-success metric, the augmented sample is automatically discarded. This filtering mechanism ensures that the augmented data maintain high task validity, providing a reliable data basis for subsequent VLA model training and simulation evaluation.

Task	Instruction	Orig.	Aug.
Tube pickup	pick_up_the_tube	30	118
Lid opening	open_centrifuge_lid	30	119
Lid closing	close_centrifuge_lid	30	120
Dish take-out	take_out_dish	30	111
Dish placement	place_dish	30	118

Table 5: Dataset statistics of five robotic manipulation tasks in the simulation platform.

G VLA Evaluation Workflow

After training two VLA models on the collected data, we evaluate their performance in the Isaac Sim simulation platform. As shown in Figure 13, the evaluation process adopts a separated server-client structure. The server loads the trained VLA policy model and receives observation inputs from the simulation environment. The client executes tasks in Isaac Sim, collects multi-view visual observations and robot states, and sends them to the server for policy inference. After receiving the predicted action from the server, the client applies the action to the robotic arm and records the execution result according to the task-specific success condition. This workflow simulates the closed-loop input-output process of VLA model execution.

Model	PR	AS	RC	Monitor Type			Monitor Param				
				Precision	Recall	F1	Precision	Recall	F1	BLEU	SciBERT
qwen3-max	✓	✓	✓	0.98±0.05	0.88±0.07	0.93±0.04	0.98±0.05	0.89±0.07	0.93±0.04	0.77±0.11	0.96±0.02
qwen3-max	×	✓	✓	0.97±0.08	0.79±0.10	0.87±0.10	0.97±0.07	0.81±0.12	0.89±0.10	0.76±0.11	0.96±0.03
qwen3-max	✓	×	✓	0.96±0.07	0.85±0.09	0.90±0.06	0.96±0.07	0.87±0.08	0.91±0.06	0.75±0.09	0.96±0.02
qwen3-max	✓	✓	×	0.96±0.07	0.82±0.10	0.89±0.06	0.96±0.06	0.83±0.09	0.89±0.05	0.73±0.09	0.95±0.02

Table 6: Monitor condition parsing results in the ablation experiment. PR, AS, and RC denote ProtoRAG, ActSchema, and RefineChecker, respectively.

Model	PR	AS	RC	Action Primitive			Action Param				Action Order	
				Precision	Recall	F1	Precision	Recall	F1	BLEU	SciBERT	L_{dn}
qwen3-max	✓	✓	✓	0.97±0.03	0.97±0.03	0.97±0.03	0.98±0.03	0.96±0.04	0.97±0.03	0.88±0.07	0.98±0.01	0.05±0.04
qwen3-max	×	✓	✓	0.96±0.04	0.96±0.03	0.96±0.03	0.97±0.04	0.96±0.05	0.97±0.04	0.85±0.09	0.98±0.02	0.07±0.05
qwen3-max	✓	×	✓	0.91±0.04	0.90±0.05	0.91±0.04	0.91±0.06	0.90±0.07	0.90±0.06	0.84±0.07	0.98±0.01	0.14±0.06
qwen3-max	✓	✓	×	0.95±0.04	0.90±0.08	0.92±0.05	0.97±0.03	0.91±0.07	0.94±0.04	0.84±0.10	0.98±0.02	0.15±0.07

Table 7: Action sequence parsing results in the ablation experiment. PR, AS, and RC denote ProtoRAG, ActSchema, and RefineChecker, respectively.

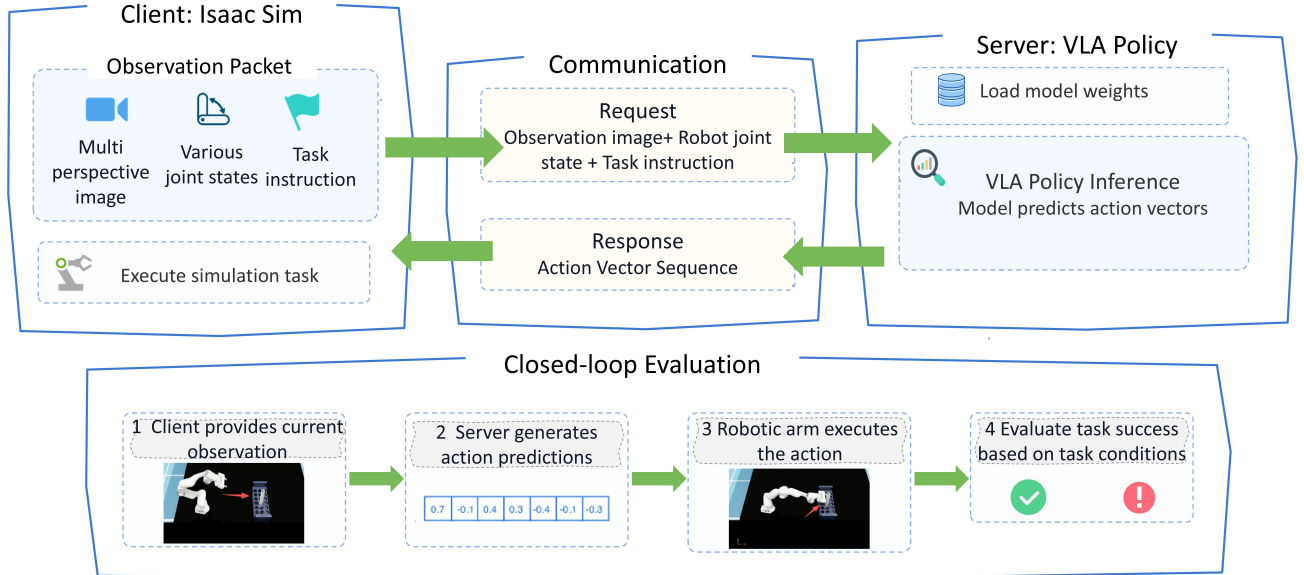


Figure 13: Online evaluation workflow of VLA models in the Isaac Sim simulation platform.