

Rethinking Intermediate Representation for VLM-based Robot Manipulation

Weiliang Tang^{1*}, Jialin Gao^{1*}, Jia-Hui Pan¹, Gang Wang¹, Li Erran Li²,
Yunhui Liu¹, Mingyu Ding³, Pheng-Ann Heng^{1†}, Chi-Wing Fu^{1†}

¹CUHK, ²Amazon, ³UNC

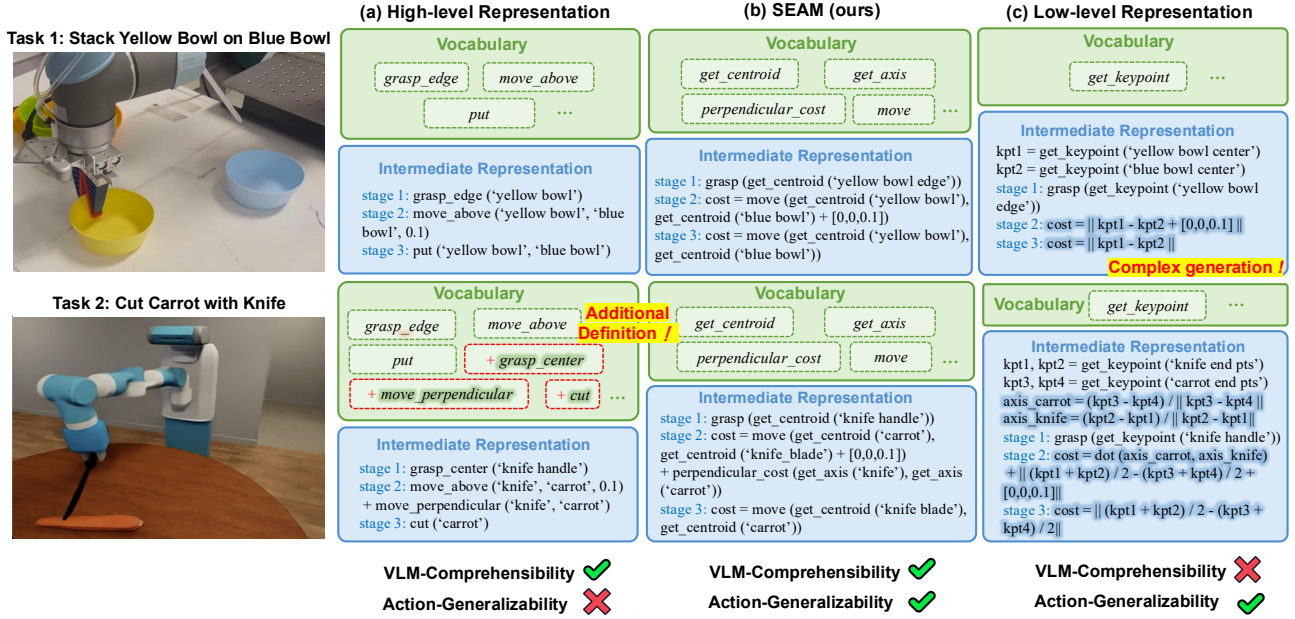


Figure 1. Comparing (a) high-level, (c) low-level, and (b) our SEAM (Semantic Assembly) representation for supporting robot manipulation. High-level representation requires manually adding new vocabulary words to customize the model for new tasks, despite its VLM-comprehensibility, whereas low-level representation requires generating complex constraints in task handling, despite its generalizability in robot actions. Our new SEAM design meets the goals of both VLM-comprehensibility and action-generalizability.

Abstract

Vision-Language Model (VLM) is an important component to enable robust robot manipulation. Yet, using it to translate human instructions into an action-resolvable intermediate representation often needs a tradeoff between VLM-comprehensibility and generalizability. Inspired by context-free grammar, we design the **Semantic Assembly** representation named SEAM, by decomposing the intermediate representation into vocabulary and grammar. Doing so leads us to a concise vocabulary of semantically-rich operations and a VLM-friendly grammar for handling diverse unseen tasks. In addition, we design a new open-vocabulary segmentation paradigm with a retrieval-augmented few-shot learning strategy to localize fine-grained object parts for manipulation, effectively with the **shortest** inference time

over all state-of-the-art parallel works. Also, we formulate new metrics for action-generalizability and VLM-comprehensibility, demonstrating the compelling performance of SEAM over mainstream representations on both aspects. Extensive real-world experiments further manifest its SOTA performance under varying settings and tasks.

1. Introduction

“The limits of my language mean the limits of my world.”
— Ludwig Wittgenstein

Leveraging the immense knowledge in Vision-Language Models (VLMs) offers a new path toward general robot manipulation. Typically, VLMs are employed to translate hu-

man instructions into an intermediate representation, specified, *e.g.*, by predefined action words and spatial constraints, such that we can solve to derive the robot actions. This approach reduces the burden of preparing extensive annotations for training vision-language action (VLA) models to align robot actions with pretrained knowledge.

Earlier VLM-based methods [2, 23, 32, 34, 35, 50] design **high-level** predefined skill words (see the example vocabularies in Fig. 1(a)) for constructing the intermediate representation. Though the skill words are easy for the VLMs to comprehend and manipulate, the construction is overly rigid. So, this approach often scales poorly to handle diverse and unseen tasks, *e.g.*, from “Stack Yellow Bowl on Blue Bowl” to “Cut Carrot with Knife”, and requires a tedious process of manually designing new skills, *i.e.*, “grasp_center”, “Cut”, and “move_perpendicular”.

Recent studies have turned to a **low-level** representation approach by considering fundamental primitives such as keypoint and axes [22, 24, 25, 41, 41, 47]); see, *e.g.*, Fig. 1(c). This approach offers greater flexibility and potential for cross-task generalization. However, the resulting intermediate representations are often overly complex, thereby difficult for VLMS to comprehend and generate, *e.g.*, the task “cutting the carrot using a grasped knife” requires ReKep [25] to generate highly specific code to explicitly calculate the constraints and costs, which is not robust. Driven by the above observations, we ask this question: *How can we create an intermediate representation that can be (i) generalizable for diverse real-world tasks (Action-generalizability) and also (ii) sufficiently understandable for VLMs to reason (VLM-comprehensibility)?*

Drawing insights from context-free grammar, we design **SEAM**, a new **Semantic Assembly** representation that connects VLM translations with robot actions by guiding the VLM to assemble words from a semantic vocabulary. The key idea is to first decompose the intermediate representation into a semantic vocabulary and a compositional grammar, jointly considering VLM readability and task scalability. Then, we guide the VLM translation by constraining the VLM to compose the vocabulary words in a semantically rich and logically coherent manner, transforming the code generation into a semantics-guided assembly process.

Specifically, *regarding VLM-comprehensibility*, the vocabulary and grammar are designed in a semantically meaningful manner for VLM-friendly translation, *e.g.*,

```
1 get_centroid(object_part_name:Str)->Point
2 move_cost(pt1:Point,pt2:Point,offset:List)->Cost
```

Regarding Action-generalizability, the vocabulary provides versatile building blocks for representing diverse actions, *e.g.*, to translate “cutting the carrot with a grasped knife” into our representation, VLM can effectively produce

```
1 perpendicular_cost(get_axis("carrot"),get_axis("
knife blade"))
```

```
2 +move_cost(get_centroid("knife"), get_centroid("
knife blade"), offset=[0,0,0.1])
```

This compositional design enables the use of a compact vocabulary that can be flexibly assembled to cater a broad range of unseen tasks. Also, to allow precise interaction with specific object parts, *e.g.*, a teapot handle or a bottle cap, we build an open-vocabulary image database, consisting of paired RGB images and segmentation masks, for fine-grained object part segmentation localization. Furthermore, we formulate a few-shot-learning-based paradigm in the retrieval-augmented generation (RAG) pipeline for accurate, instruction-specified part segmentation.

To evaluate the effectiveness of SEAM, we design two quantitative metrics, namely *action-generalizability* and *VLM-comprehensibility*. The experimental results manifest that SEAM is able to achieve strong performance on both metrics. Also, it largely improves real-world robot manipulation performance by 15% over prior SOTA approaches, showing its superior robustness for practical usage.

Below, we summarize the main contributions:

- We design the new Semantic Assembly representation (SEAM) for VLM-based robot manipulation, bridging the gap between VLM reasoning and robot action execution.
- We introduce an RAG-based few-shot learning segmentation pipeline, enabling efficient, fine-grained, object-part level open-vocabulary segmentation.
- To our best knowledge, this is the first study on analyzing the intermediate representation for VLM-based robot manipulation by formulating two new metrics, VLM-comprehensibility and action-generalizability. Extensive experiments demonstrate the compelling performance of SEAM over prior SOTAs by 15%.

2. Related Work

Foundation Model for Robot Manipulation: Recent research on robot manipulation often relies on foundation models. In general, there are two main types: (i) VLA-based methods [5–9, 11, 17, 26, 37] finetune a pretrained VLM end-to-end with action data to generate robot actions in a see-and-act manner. Despite its high-frequency control and simple design, it requires massive labeled data for training. (ii) VLM-based methods [2, 23, 32–34, 39, 45, 46, 51] bridge human instructions and robot actions via an intermediate language. Some works design high-level semantic APIs as the intermediate representation and harness a VLM to decompose tasks into a sequence of API calls. Some other works harness the spatial relation of objects as the intermediate representation, *e.g.*, keypoints [25], manipulation primitive [41], and task-relevant 3D component [22]. Another series of works [15, 24, 40, 47] employ a VLM to directly generate code to derive the robot actions.

Open-Vocabulary Object Detection and Part Segmen-

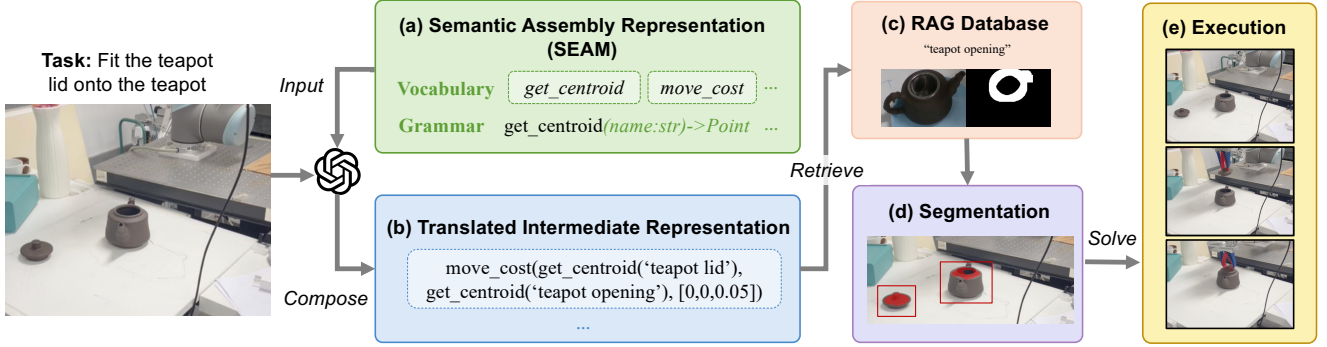


Figure 2. Overall pipeline of our method. Given the current observation and the task instruction, our method first generates the (a) Semantic Assembly Representation (SEAM) with designed vocabulary and grammar, and then (b) translated into an intermediate representation. Next, we retrieve the corresponding support images and support masks from (c) the Retrieval Augmented Generation (RAG) Database and (d) segment the target object parts in the scene. Finally, we solve the gripper’s trajectories for (e) robotic execution.

tation. Open-vocabulary object detection and open-vocabulary part segmentation are challenging problems, due to training data scarcity and semantic complexity. Regarding open-vocabulary object detection, some works [12, 14, 18, 28, 54, 60] attempt to align text embedding with image embedding in the latent space, whereas others [1, 28, 55] finetune a pretrained backbone on a large amount of visual data and harness the extracted visual-semantic features. Some more recent works [4, 16, 31] propose to finetune a VLM with object detection data to harness the VLM’s ability on spatial and semantic understanding.

Regarding open-vocabulary part segmentation, some works [47, 57, 59, 61] suggest various mechanisms to prompt the class-agnostic segmentation model, *e.g.*, SAM [27], to directly predict the mask. Some others [29, 49, 52, 58, 62] finetune a VLM with segmentation data to align the mask generation with the pretrained semantics. However, satisfactory performance is hard to achieve in segmenting object parts for robot manipulation.

Automata Theory for Robot Manipulation. We observe a strong connection between the design of the intermediate representation and early works in modeling and controlling robots with automata theory. System modeling with discrete, event-driven logic is a well-established field. The first work dates back to [43], in which the automata theory [42] is utilized to model an events system. Later, hybrid automaton [3, 10, 20, 21, 38] combines a Finite Automaton with differential equations. More recently, [13] proposes Motion Grammar to model robot manipulation with context-free grammar and discusses the completeness, correctness, and decidability for manipulation tasks. Notations in our work are inspired by context-free grammar. For better readability, we use the term vocabulary to collectively refer to lexicon, grammar for rules, and sacrifice rigorous formulation, without loss of generality.

3. Method

In this section, we first present the problem formulation (Section 3.1). Then, we explain the relationships between SEAM and context-free grammar (Section 3.2), elaborate on the design of SEAM (Section 3.3), present our novel open-vocabulary segmentation paradigm (Section 2), and lastly discuss how we convert the intermediate representation to robot actions (Section 3.5). The overall pipeline of our method is illustrated in Fig. 2.

3.1. Problem Formulation

Given a human instruction L and visual input I , the VLM-based robot manipulation task can be formulated as a translation process: $\mathcal{R} = \text{VLM}(L, I)$, where $\mathcal{R}$ is an intermediate representation, which can be passed into a solver to generate the robot actions, such as the gripper’s 3D position $\{x, y, z\}$, 3D orientation (*e.g.*, Euler angles $\{X, Y, Z\}$), and gripper status O . We formulate $\mathcal{R}$ as a linguistic representation: $\bar{\mathcal{R}} = (\mathcal{V}, \mathcal{G})$, where $\mathcal{V}$ is the vocabulary that comprises a set of skill words and $\mathcal{G}$ is the grammar that specifies the rules for their valid composition. The core challenge lies in the design of the skill words in the vocabulary and rules in the grammar, to make them amenable for VLM to generate, while being expressive to facilitate zero-shot generalization to handling diverse and unseen tasks.

3.2. Context-Free Grammar and SEAM

A Context-Free Grammar (CFG) consists of a finite set of symbols (words) and a finite set of recursive production rules, that together enable us to model a language and generate string patterns within the language space. Formally, it can be defined as a 4-tuple (V, Σ, R, S) , which includes a set of non-terminal symbols V , terminal symbols Σ , production rules R that map non-terminals to sequences of terminals and non-terminals, and a start symbol S . CFG provides a structured yet flexible framework for defining the

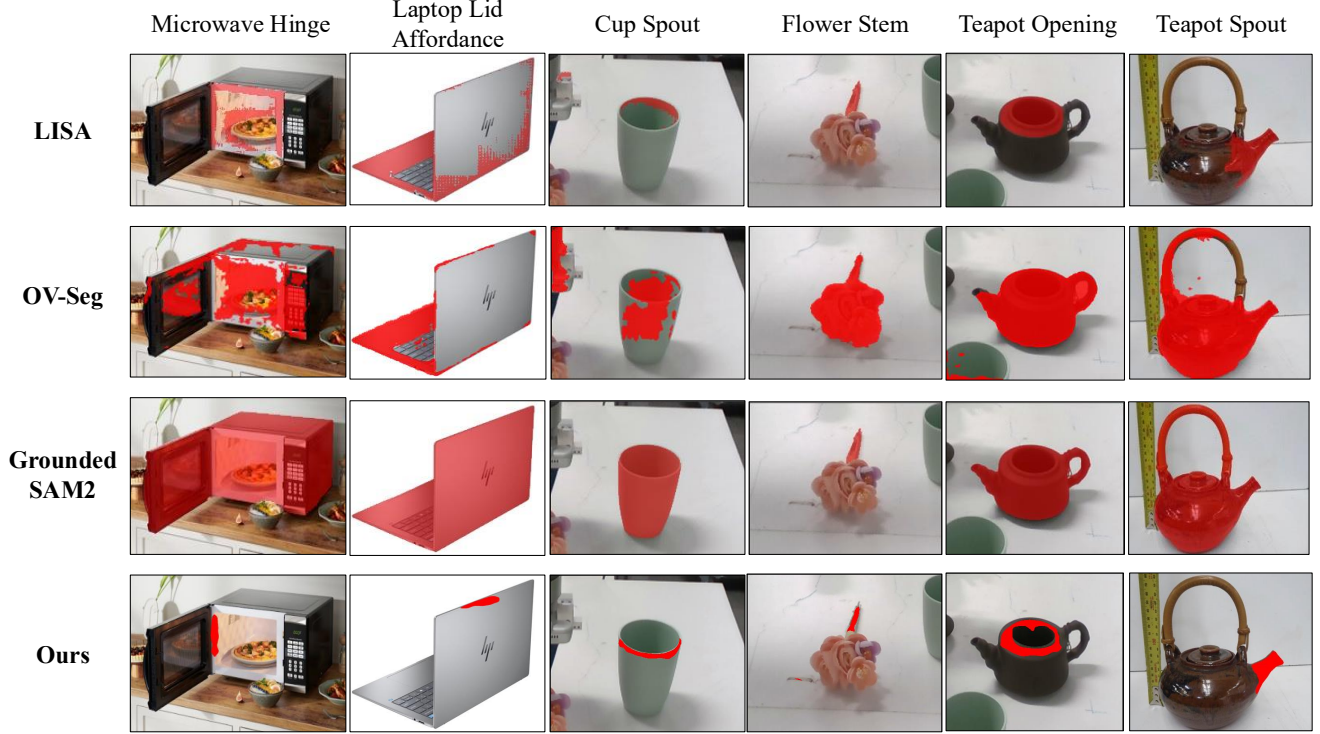


Figure 3. Qualitative performance comparisons for open vocabulary segmentation between the state-of-the-art methods and our methods on common manipulation.

Table 1. The design of SEAM composes of vocabulary $\mathcal{V}$ and grammar $\mathcal{G}$ listed below.

Vocabulary $\mathcal{V}$	Grammar $\mathcal{G}$
get_axis, rotate_cost,	cost $\rightarrow$ cost + cost,
get_centroid, orbit_cost,	get_axis,object $\rightarrow$ vec,
get_height, gripper_close,	get_centroid,object $\rightarrow$ pt,
move_cost, gripper_open,	get_height,object $\rightarrow$ cost,
parallel_cost,	parallel_cost,vec,vec $\rightarrow$ cost,
get_gripper_pos	move_cost,pt,pt $\rightarrow$ cost,
perpendicular_cost,	pt $\rightarrow$ pt $\pm$ pt

syntax of formal languages, enabling unambiguous parsing and interpretation of complex symbolic expressions.

We adapt the powerful CFG framework to design SEAM and to create a more intuitive and semantically-grounded interface for VLM, as shown in Tab. 1. SEAM leverages the structural rigor of a CFG but is specifically designed for VLM comprehensibility. The SEAM’s vocabulary $\mathcal{V}$ is parallel to a union of sets V , Σ , and S in CFG, yet offering semantically-rich, human-readable words for modeling robot manipulation tasks. The SEAM’s grammar $\mathcal{G}$ is parallel to the production rules R in CFG, yet the rules are formatted in a semantically-rich and human-readable manner.

3.3. SEAM Representation Design

We elaborate on the design of our representation here. Specifically, we will design our vocabulary $\mathcal{V}$ and grammar $\mathcal{G}$. Our ultimate goal is to balance the VLM-comprehensibility and action-generalizability. In order to achieve this, we stick to the following principles:

- **VLM-Readability:** Vocabulary words in $\mathcal{V}$ are grounded in the VLM’s semantic space for accurate comprehension.
- **Proper Abstraction:** The representation abstracts away implementation details, exposing only the parameters essential for task planning and generalization.
- **Conciseness:** Semantics are conveyed through a minimal sequence of symbols, using high-level words to maximize information density.
- **Reliability:** A type system within $\mathcal{G}$ constrains the composition of $\mathcal{V}$, ensuring syntactically valid and semantically sound output from the VLM.
- **Proper Minimalism:** The core words are kept small and intuitive to reduce the VLM’s learning burden and prevent unexpected behaviors.
- **Composability:** The representation is inherently modular, supporting extensibility through the seamless integration of new primitives and rules.

Our design helps to achieve good VLM-readability, as

all the words in the vocabulary are rich in semantics and close to human natural language. We achieve good abstraction since all the words in the vocabulary abstract away the repetitive and tedious underline implementation (e.g., using principal components analysis to calculate the axis in “get_axis”), leaving only necessary parameters exposed (e.g., the points as input in “get_axis”). Our design aims to make vocabulary orthogonal, that is, to minimize semantic overlap among elements, thereby achieving conciseness. The grammar rules in $\mathcal{G}$ define the data type for vocabulary assembly, which constrains the VLM to assemble the vocabulary according to the grammar rules and produce more reliable outputs. By keeping our vocabulary and grammar rules minimum, the VLM can be more focused in distinguishing words and makes the output more predictable. Though minimal, our words in the vocabulary can assemble a wide range of actions via the grammar rules, showcasing our design’s strong extensibility.

3.4. RAG-based Few-shot Open-vocabulary Segmentation

Grounding target object parts (e.g., teapot opening, flower stem) on words such as “get_centroid” and “get_axis” in $\mathcal{V}$ requires a precise and fine-grained open-vocabulary method. As Fig. 3 shows, existing state-of-the-art segmentation models struggle to achieve satisfactory performance. For example, OV-Seg [53] and Grounded SAM2 [44] tend to segment the whole objects/parts rather than associated parts for robot interaction, whereas LISA [29] falls short of segmenting the hinge, opening, and affordance. To support robot manipulation with fine-grained object part segmentation, we build a database $\mathcal{D} = \{(\mathcal{K}_i, \mathcal{P}_i)\}_{i=1}^N$, where $\mathcal{K}$ is a set of key phrases that describe specific object parts, e.g., the key phrases of the object part “cup opening” can be {cup opening, cup rim, cup edge}; and $\mathcal{P}$ is a set of support pairs defined as $\mathcal{P} = \{(I_j^S, M_j^S)\}_{j=1}^n$, where I_j^S is the j -th support image containing the object part and M_j^S is the associated binary support mask that highlights the object part in I_j^S . Given a query image I_Q and a natural language instruction $desc$, we retrieve all support pairs $\mathcal{P}_i$ with the matched key phrase $\mathcal{K}_i$. Specifically, we calculate the Levenshtein distance [30] between $desc$ and each key phrase in $\mathcal{K}_i$, and target $\mathcal{K}_i$ contains a key phrase with the least Levenshtein distance to $desc$. The details of our database for RAG-based open-vocabulary segmentation are provided in the Appendix.

In detail, during the generation phase, our target is to generate query mask M_Q based on the retrieved $\mathcal{P}_i$. Here, we adopt a few-shot segmentation network as Mapper [48], which essentially maps the support masks M_S to the query mask M_Q , according to the support feature and query feature similarity calculated as the attention score.

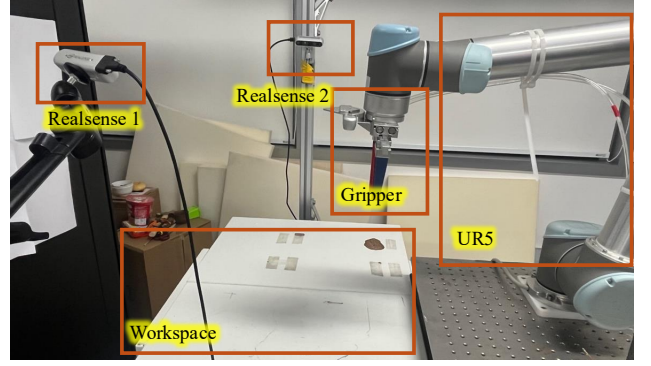


Figure 4. Our system setup includes a robotic module (A UR5 robot equipped with a gripper) and a visual perception module (two realsense D435 cameras on opposite side of the workspace).

3.5. Trajectory Generation

To convert our intermediate representation to robot actions, we solve for the gripper end pose to minimize the overall costs evaluated by our intermediate representation. Specifically, the SEAM representation is Python-executable, and the execution result is a numerical cost that captures point cloud P matches the SEAM representation. Next, we locate which object parts are moving along with the gripper by identifying whether they belong to the grasped object. The point clouds of these moving object parts are denoted as P^m , while stationary object parts are represented as P^s .

Since the gripper is rigidly attached to the point cloud of the moving parts P^m , they should share the same transformation. Thus, solving for the gripper’s target rotation $\mathbf{R} \in SO(3)$ and translation $\mathbf{t} \in \mathbb{R}^3$ reduces to the following optimization problem:

$$\min_{\mathbf{R}, \mathbf{t}} \text{language} (P^s \cup (\mathbf{R}\mathbf{R}_0^{-1}(P^m - \mathbf{t}_0) + \mathbf{t})) + \alpha \|\mathbf{t} - \mathbf{t}_0\|_2 + \beta \|\text{euler}(\mathbf{R}\mathbf{R}_0^{-1})\|_1, \quad (1)$$

where $\mathbf{R}_0$ and $\mathbf{t}_0$ represent the gripper’s initial pose, $\text{euler}(\cdot)$ extracts the Euler angles. The last two terms are the regularization terms with α, β as the two weights to encourage the gripper moves with the least transition and rotation, respectively.

4. Experiment

4.1. Experimental Setup

Hardware Fig. 4 depicts our hardware setup, in which our robotic platform is built using the UR5 industrial robot with a gripper as its end-effector. To obtain visual perception, we set up a dual-camera module by arranging two Intel RealSense D435 depth-sensing cameras, strategically positioned on the opposite sides of the workspace. Doing so enables us to obtain a comprehensive stereo coverage of the operational area for more robust 3D scene perception.

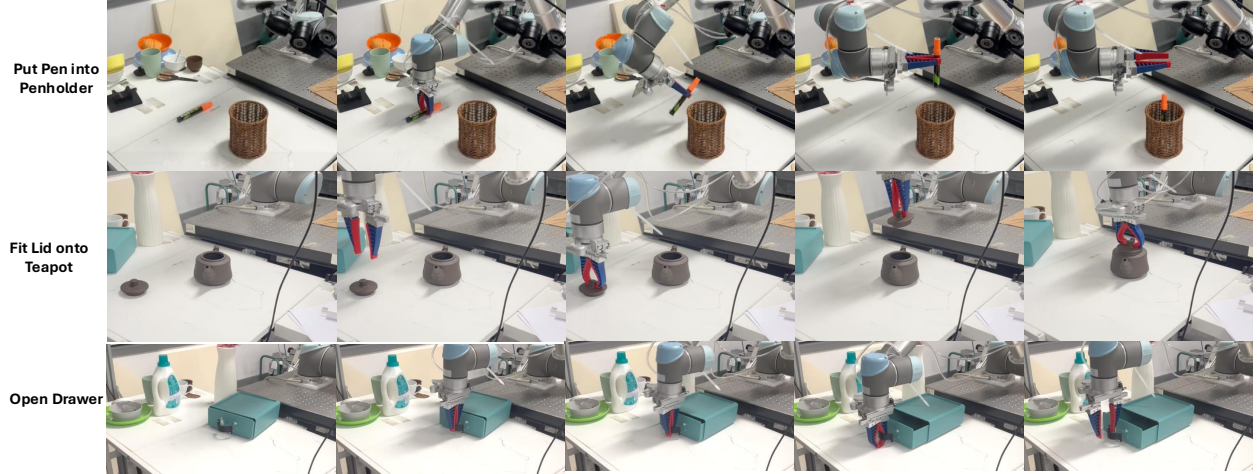


Figure 5. Execution sequences of the three tasks in the real-world environment. The sequences of all the tasks can be found in the Appendix.

Implementation Details We deployed Qwen3-VL-30B-22A [56] on an A100 GPU as the VLM in our method. Regarding the RAG-based open-vocabulary segmentation, we adopt the Swin-B transformer [36] for feature extraction and the trained matcher from [48].

4.2. Reference Prompt for Intermediate Representation Generation

Similar to our comparison methods [22, 24, 25, 41], to improve the robustness of intermediate representation generation, we build a prompt for the VLM models to refer to. The prompt consists of intermediate representation for some “atomic action”. As an example, for the action “move something relative to something”, the intermediate representation is “move_cost([something], [something], offset)”. These templates provide strong hint to prevent the VLM from hallucination or generating incomplete representation. The complete prompt for “atomic action” and the corresponding representation are listed in the Appendix.

4.3. Evaluation Tasks and Performance Measures

We developed a suite of eight distinct activities to test robotic manipulation performance under realistic conditions. The benchmark includes six tasks requiring interaction with rigid objects (such as pouring beverages) and six additional tasks involving articulated mechanisms (like drawer opening). This diverse selection incorporates various object types to examine generalization capacity and adaptation in challenging settings. Each method underwent 10 repeated trials per task with success rates documented as the primary metric. To prevent evaluation bias, we randomized object configurations between trials.

4.4. Comparison Methods

Our method is evaluated against four established methods:

VoxPoser [24] synthesizes robot motion trajectories by constructing 3D value maps through large language and vision-language models, demonstrating strong zero-shot reasoning and continuous feedback capabilities;

CoPa [22] incorporates component-level spatial limitations integrated with visual-language understanding to facilitate manipulation without vocabulary restrictions;

ReKep [25] implements relationship-based keypoint constraints and uses multi-level optimization to translate natural language commands into immediate action sequences.

OmniManip [41] uses the manipulation-aware primitives to construct spatial constraints for action solving.

4.5. Main Results

We conduct experiments on eight tasks and the success rates are illustrated in Tab. 2. Following [41], we have two variants with close-loop and open-loop planning. The objects poses and positions are randomly initialized. We can conclude that our method achieves consistent better zero-shot generalizability across wide ranges of tasks with different objects poses and positions, surpassing the OmniManip by 15% in the average success rates. The execution sequences of tasks are illustrated in Fig. 5. The complete execution sequences for the eight tasks are listed in the Appendix. Next, we analyze how to achieve the performance with our two key designs: (i) SEAM, and (ii) RAG-based open-vocabulary segmentation.

Semantic Assembly Representation Our SEAM allows VLM to comprehend and compose the intermediate representation. As illustrated in Fig. 6, for the task “put the pen

Table 2. Performance comparison across different methods

Tasks	Methods						
	VoxPoser	CoPa	ReKep	OmniManip		SEAM (ours)	
				Closed-loop	Open-loop	Closed-loop	Open-loop
Insert the pen in holder	0/10	4/10	3/10	7/10	5/10	8/10	6/10
Recycle the battery	6/10	5/10	7/10	8/10	6/10	8/10	6/10
Pick up cup/bowl onto the dish	3/10	2/10	9/10	8/10	7/10	9/10	5/10
Fit the lid onto the teapot	0/10	2/10	3/10	5/10	3/10	7/10	7/10
Open the drawer	1/10	4/10	-	6/10	4/10	8/10	5/10
Close the drawer	3/10	3/10	-	8/10	6/10	9/10	5/10
Press the red button	0/10	3/10	-	7/10	6/10	10/10	9/10
Open the jar	2/10	0/10	-	6/10	5/10	8/10	8/10
Total	18.6%	28.8%	-	68.8%	52.5%	83.8%	63.8%

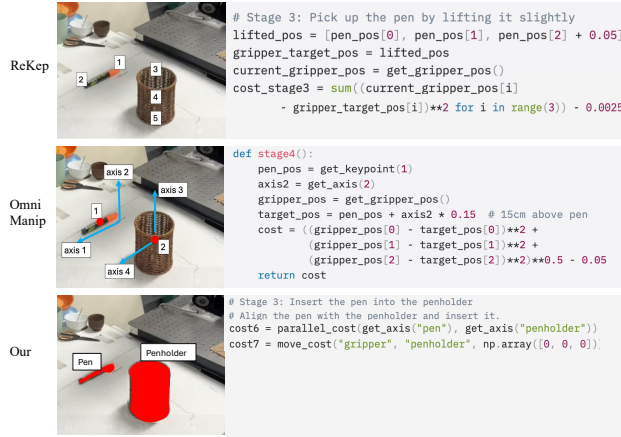


Figure 6. Comparisons among the intermediate representation generated by state-of-the-art methods and our methods for the task “put the pen into the penholder”.

into the penholder”, in the stage “move the pen above the penholder”, ReKep [25] struggles to align the pen’s axis with the penholder’s axis, requires the VLM to solely use “get_keypoint” to select the correct keypoints and calculate the axis from the labeled image. OmniManip [41] also struggles to align the pen’s axis with the penholder’s axis, which requires the VLM to select from the image the correct axis and compose the correct representation. In contrast, our SEAM abstracts away the complications of direction alignment with semantic-rich vocabulary and grammar, so it can easily generate the correct representation to align the pen and the penholder’s axis.

RAG-based Open-Vocabulary Segmentation The RAG-based open-vocabulary segmentation pipeline enables us to flexibly locate any object part essential for manipulation. We qualitatively compare the performance of our methods with existing SOTA methods in Fig. 3. More segmentation comparison results can be found in the Appendix. In our real-world experiment, our segmentation method to

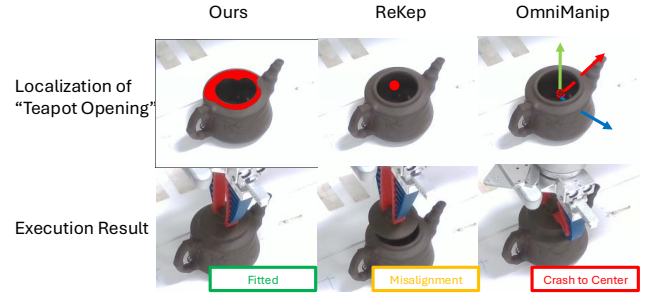


Figure 7. Comparisons among open-vocabulary localization for “teapot opening” and their corresponding execution results in the task “fit the teapot lid on the teapot”.

Table 3. Comparisons of time elapse between our method and the state-of-the-art methods.

Model	LISA	OV-seg	Grounded SAM	SEAM (ours)
Time (sec.)	0.9	10.2	0.88	0.6

a large extent helps improve the success rates on tasks requiring careful alignment. For example, in the task “fit the lid on the teapot”, we successfully locate the rim of the teapot opening, which provides us with an accurate location to move the lid down, as illustrated in Fig. 7. Other methods can only locate the teapot center [24, 32, 41] or the interior point inside the teapot [25] as illustrated in Fig. 7, which may lead to misalignment or crushing the lid into the teapot’s center. Furthermore, we deploy all the methods on an A6000 GPU to compare time elapse. The results in Tab. 3 demonstrate the efficiency of our method.

4.6. Quantitative Study on Action-Generalizability and VLM-Comprehensibility

In this section, we compare the performance of existing mainstream VLM-based robot manipulation methods on the design of intermediate representation, and prove our Se-

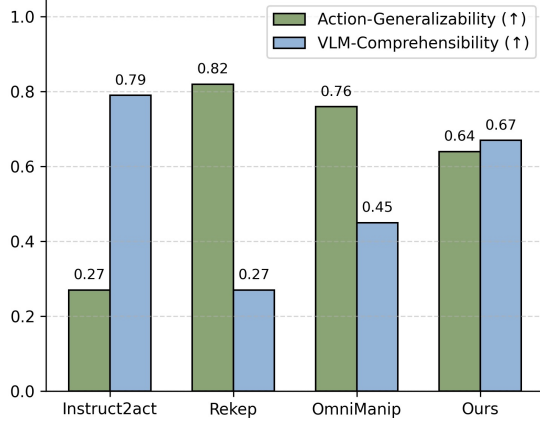


Figure 8. Statistics on each method’s action-generalizability and VLM-comprehensibility.

mantic Assembly Representation (SEAM) is better.

Evaluation Metrics. To evaluate the performance of intermediate representation, we focus on two metrics: (i) **Action-generalizability**: How well the intermediate representation can generalize to unseen manipulation tasks. (ii) **VLM-comprehensibility**: With the spatial reasoning and semantic understanding of the existing VLMs, whether the intermediate representation can be correctly generated given the action instruction. Formally, action-generalizability (AG) is defined as:

$$AG = 1 - \frac{|\mathcal{V}|}{T},$$

where $|\mathcal{V}|$ denotes the number of unique vocabulary operations required to translate all human instructions I , and T is the total number of manipulation tasks. A higher action-generalizability indicates that fewer vocabulary operations are needed to represent all tasks, implying stronger generalization ability. VLM-comprehensibility (VC) measures the proportion of tasks that a VLM can successfully complete given human instructions, which is formulated as:

$$VC = \frac{N_{\text{succ}}}{T},$$

where N_{succ} is the number of successfully executed tasks.

Baselines. We compare SEAM on action-generalizability and VLM-comprehensibility with one high-level representation method: **Instruct2Act** [23] uses predefined action APIs to construct intermediate representation, and two low-level representation methods: (i) **ReKep** [25] adopts keypoints for intermediate representation. (ii) **Omnimanip** [41] uses intermediate representation based on axis and keypoint primitives. We also summarize the vocabulary $\mathcal{V}$ for each of the methods, partly listed in Tab. 4. The complete vocabulary $\mathcal{V}$ and grammar is $\mathcal{G}$ listed in Appendix.

Table 4. $\mathcal{V}$ and $\mathcal{G}$ for each method.

Method	Vocabulary $\mathcal{V}$
ReKep	get_keypoint, move_to, $\mathcal{V}_{\text{Python}}$
OmniManip	get_keypoint, get_axis, move_to,
Instruct2act	find, pick, place, pick_place, insert, push, press, flip, scoop, stir, tap, fold, straighten, move_above, move_perpendicular, route_around, move_parallel, screw_rotation, controlled_pour, $\mathcal{V}_{\text{Python}}$

Task. We randomly generate 33 manipulation tasks on single-arm, non-tactile and non-force-feedback. For each method, we prompt Qwen3-VL [56] with the corresponding $\mathcal{V}$ and $\mathcal{G}$ to generate intermediate representation. Since it’s laborious-infeasible to validate the success rate with a real or virtual environment for all the tasks, we employ DeepSeek [19] to help evaluate whether the intermediate representation can reasonably produce actions that complete the tasks successfully. The implementation details can be found in the Appendix.

Experiment Results. The action generalizability and the VLM-comprehensibility results are demonstrated in Fig. 8 From the experimental results, we can clearly validate: (i) There is a trade-off between the action-generalizability and VLM-comprehensibility. Previous state-of-the-art methods’ intermediate representations are either too high-level (high VLM-comprehensibility, low action-generalizability) [32] or too low-level (low VLM-comprehensibility, high action-generalizability) [25, 41]. (ii) Our SEAM balances the VLM-comprehensibility and action-generalizability.

5. Conclusion

In this work, we observe that there is a tradeoff between the VLM-comprehensibility and the action-generalizability among the intermediate representations chosen by various VLM-based robot manipulation methods. To address this, we create Semantic Assembly Representation (SEAM) by designing our vocabulary and grammar stick to our principles. To localize the object part for representation evaluation, we design a RAG-based open-vocabulary segmentation method which can efficiently and effectively segment fine-grained object parts. Additionally, We introduce two metrics to demonstrate the scalability and comprehensibility of VLMs for our SEAM. Our analytical experiment results show that our method can generate intermediate representation robustly and achieve state-of-the-art performance in the real-world experiments.

References

- [1] Cora: Adapting clip for open-vocabulary detection with region prompting and anchor pre-matching, 2023. 3
- [2] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022. 2
- [3] Rajeev Alur, Costas Courcoubetis, Thomas A Henzinger, and Pei-Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *International hybrid systems workshop*, pages 209–229. Springer, 1991. 3
- [4] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibao Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025. 3
- [5] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. *pi*-0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. 2
- [6] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y Galliker, et al. π 0.5: a vision-language-action model with open-world generalization. In *9th Annual Conference on Robot Learning*, 2025.
- [7] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.
- [8] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.
- [9] Qingwen Bu, Jisong Cai, Li Chen, Xiuqi Cui, Yan Ding, Siyuan Feng, Shenyuan Gao, Xindong He, Xuan Hu, Xu Huang, et al. Agibot world colosseum: A large-scale manipulation platform for scalable and intelligent embodied systems. *arXiv preprint arXiv:2503.06669*, 2025. 2
- [10] Christos G Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer, 2007. 3
- [11] Chi-Lam Cheang, Guangzeng Chen, Ya Jing, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Hongtao Wu, Jiafeng Xu, Yichu Yang, et al. Gr-2: A generative video-language-action model with web-scale knowledge for robot manipulation. *arXiv preprint arXiv:2410.06158*, 2024. 2
- [12] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xingang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16901–16911, 2024. 3
- [13] Neil Dantam and Mike Stilman. The motion grammar: Linguistic perception, planning, and control. *Robotics: science and systems VII*, pages 49–56, 2012. 3
- [14] Yu Du, Fangyun Wei, Zihe Zhang, Miaojing Shi, Yue Gao, and Guoqi Li. Learning to prompt for open-vocabulary object detection with vision-language model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14084–14093, 2022. 3
- [15] Jiafei Duan, Wentao Yuan, Wilbert Pumacay, Yi Ru Wang, Kiana Ehsani, Dieter Fox, and Ranjay Krishna. Manipulate-anything: Automating real-world robots using vision-language models. *arXiv preprint arXiv:2406.18915*, 2024. 2
- [16] Shenghao Fu, Qize Yang, Qijie Mo, Junkai Yan, Xihan Wei, Jingke Meng, Xiaohua Xie, and Wei-Shi Zheng. Llm-det: Learning strong open-vocabulary object detectors under the supervision of large language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 14987–14997, 2025. 3
- [17] Chongkai Gao, Zixuan Liu, Zhenghao Chi, Junshan Huang, Xin Fei, Yiwen Hou, Yuxuan Zhang, Yudi Lin, Zhirui Fang, Zeyu Jiang, et al. Vla-os: Structuring and dissecting planning representations and paradigms in vision-language-action models. *arXiv preprint arXiv:2506.17561*, 2025. 2
- [18] Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. Open-vocabulary object detection via vision and language knowledge distillation. *arXiv preprint arXiv:2104.13921*, 2021. 3
- [19] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. 8
- [20] Thomas A Henzinger. The theory of hybrid automata. In *Proceedings 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292. IEEE, 1996. 3
- [21] Dimitrios Hristu-Varsakelis and William S Levine. *Handbook of networked and embedded control systems*. Springer, 2005. 3
- [22] Haoxu Huang, Fanqi Lin, Yingdong Hu, Shengjie Wang, and Yang Gao. Copa: General robotic manipulation through spatial constraints of parts with foundation models. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9488–9495. IEEE, 2024. 2, 6
- [23] Siyuan Huang, Zhengkai Jiang, Hao Dong, Yu Qiao, Peng Gao, and Hongsheng Li. Instruct2act: Mapping multi-modality instructions to robotic actions with large language model. *arXiv preprint arXiv:2305.11176*, 2023. 2, 8
- [24] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023. 2, 6, 7
- [25] Wenlong Huang, Chen Wang, Yunzhu Li, Ruohan Zhang, and Li Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024. 2, 6, 7, 8
- [26] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, Quan Vuong, Thomas Koliar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey

- Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. 2
- [27] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023. 3
- [28] Weicheng Kuo, Yin Cui, Xiuye Gu, AJ Piergiovanni, and Anelia Angelova. F-vlm: Open-vocabulary object detection upon frozen vision and language models. *arXiv preprint arXiv:2209.15639*, 2022. 3
- [29] Xin Lai, Zhuotao Tian, Yukang Chen, Yanwei Li, Yuhui Yuan, Shu Liu, and Jiaya Jia. Lisa: Reasoning segmentation via large language model. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9579–9589, 2024. 3, 5
- [30] VI Lcvnshtcin. Binary coors capable or ‘correcting deletions, insertions, and reversals. In *Soviet physics-doklady*, 1966. 5
- [31] Jincheng Li, Chunyu Xie, Ji Ao, Dawei Leng, and Yuhui Yin. Lmm-det: Make large multimodal models excel in object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 308–318, 2025. 3
- [32] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500. IEEE, 2023. 2, 7, 8
- [33] Fangchen Liu, Kuan Fang, Pieter Abbeel, and Sergey Levine. Moka: Open-vocabulary robotic manipulation through mark-based visual prompting. In *First Workshop on Vision-Language Models for Navigation and Manipulation at ICRA 2024*, 2024.
- [34] Siyuan Liu, Jiawei Du, Sicheng Xiang, Zibo Wang, and Dingsheng Luo. Relep: A novel framework for real-world long-horizon embodied planning. *arXiv e-prints*, pages arXiv–2409, 2024. 2
- [35] Yibin Liu, Zhixuan Liang, Zanzin Chen, Tianxing Chen, Mengkang Hu, Wanxi Dong, Congsheng Xu, Zhaoming Han, Yusen Qin, and Yao Mu. Hycodpolicy: Hybrid language controllers for multimodal monitoring and decision in embodied agents. *arXiv preprint arXiv:2508.02629*, 2025. 2
- [36] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021. 6
- [37] Qi Lv, Weijie Kong, Hao Li, Jia Zeng, Zherui Qiu, Delin Qu, Haoming Song, Qizhi Chen, Xiang Deng, and Jiangmiao Pang. F1: A vision-language-action model bridging understanding and generation to actions. *arXiv preprint arXiv:2509.06951*, 2025. 2
- [38] John Lygeros, Karl Henrik Johansson, Slobodan N Simic, Jun Zhang, and S Shankar Sastry. Dynamical properties of hybrid automata. *IEEE Transactions on automatic control*, 48(1):2–17, 2003. 3
- [39] Zhao Mandi, Shreeya Jain, and Shuran Song. Roco: Dialectic multi-robot collaboration with large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 286–299. IEEE, 2024. 2
- [40] Yao Mu, Juntong Chen, Qinglong Zhang, Shoufa Chen, Qiaojun Yu, Chongjian Ge, Runjian Chen, Zhixuan Liang, Mengkang Hu, Chaofan Tao, et al. Robocodex: Multimodal code generation for robotic behavior synthesis. *arXiv preprint arXiv:2402.16117*, 2024. 2
- [41] Mingjie Pan, Jiyao Zhang, Tianshu Wu, Yinghao Zhao, Wenlong Gao, and Hao Dong. Omnimanip: Towards general robotic manipulation via object-centric interaction primitives as spatial constraints. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 17359–17369, 2025. 2, 6, 7, 8
- [42] Michael O Rabin and Dana Scott. Finite automata and their decision problems. *IBM journal of research and development*, 3(2):114–125, 1959. 3
- [43] Peter J Ramadge and W Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987. 3
- [44] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, et al. Grounded sam: Assembling open-world models for diverse visual tasks. *arXiv preprint arXiv:2401.14159*, 2024. 5
- [45] Yide Shentu, Philipp Wu, Aravind Rajeswaran, and Pieter Abbeel. From llms to actions: Latent codes as bridges in hierarchical robot control. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8539–8546. IEEE, 2024. 2
- [46] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M Sadler, Wei-Lun Chao, and Yu Su. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009, 2023. 2
- [47] Weiliang Tang, Jia-Hui Pan, Yun-Hui Liu, Masayoshi Tomizuka, Li Erran Li, Chi-Wing Fu, and Mingyu Ding. Geomanip: Geometric constraints as general interfaces for robot manipulation. *arXiv preprint arXiv:2501.09783*, 2025. 2, 3
- [48] Weiling Tang, Biqi Yang, Pheng-Ann Heng, Yun-Hui Liu, and Chi-Wing Fu. Overcoming support dilution for robust few-shot semantic segmentation. *arXiv preprint arXiv:2501.13529*, 2025. 5, 6
- [49] Junchi Wang and Lei Ke. Llm-seg: Bridging image segmentation and large language model reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1765–1774, 2024. 3
- [50] Yuki Wang, Gonzalo Gonzalez-Pumariaga, Yash Sharma, and Sanjiban Choudhury. Demo2code: From summarizing demonstrations to synthesizing code via extended chain-of-thought. *Advances in Neural Information Processing Systems*, 36:14848–14956, 2023. 2
- [51] Yongdong Wang, Runze Xiao, Jun Younes Louhi Kasahara, Ryosuke Yajima, Keiji Nagatani, Atsushi Yamashita, and Hajime Asama. Dart-llm: Dependency-aware multi-robot

task decomposition and execution using large language models. *arXiv preprint arXiv:2411.09022*, 2024. 2

- [52] Meng Wei, Xiaoyu Yue, Wenwei Zhang, Shu Kong, Xihui Liu, and Jiangmiao Pang. Ov-parts: Towards open-vocabulary part segmentation. *Advances in Neural Information Processing Systems*, 36:70094–70114, 2023. 3
- [53] Meng Wei, Xiaoyu Yue, Wenwei Zhang, Shu Kong, Xihui Liu, and Jiangmiao Pang. Ov-parts: Towards open-vocabulary part segmentation. *Advances in Neural Information Processing Systems*, 36, 2024. 5
- [54] Size Wu, Wenwei Zhang, Sheng Jin, Wentao Liu, and Chen Change Loy. Aligning bag of regions for open-vocabulary object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 15254–15264, 2023. 3
- [55] Size Wu, Wenwei Zhang, Lumin Xu, Sheng Jin, Xiangtai Li, Wentao Liu, and Chen Change Loy. Clipself: Vision transformer distills itself for open-vocabulary dense prediction. *arXiv preprint arXiv:2310.01403*, 2023. 3
- [56] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 6, 8
- [57] Haobo Yuan, Xiangtai Li, Chong Zhou, Yining Li, Kai Chen, and Chen Change Loy. Open-vocabulary sam: Segment and recognize twenty-thousand classes interactively. In *European Conference on Computer Vision*, pages 419–437. Springer, 2024. 3
- [58] Haobo Yuan, Xiangtai Li, Tao Zhang, Zilong Huang, Shilin Xu, Shunping Ji, Yunhai Tong, Lu Qi, Jiashi Feng, and Ming-Hsuan Yang. Sa2va: Marrying sam2 with llava for dense grounded understanding of images and videos. *arXiv preprint arXiv:2501.04001*, 2025. 3
- [59] Renrui Zhang, Zhengkai Jiang, Ziyu Guo, Shilin Yan, Junting Pan, Xianzheng Ma, Hao Dong, Peng Gao, and Hongsheng Li. Personalize segment anything model with one shot. *arXiv preprint arXiv:2305.03048*, 2023. 3
- [60] Yiwu Zhong, Jianwei Yang, Pengchuan Zhang, Chunyuan Li, Noel Codella, Liunian Harold Li, Luwei Zhou, Xiyang Dai, Lu Yuan, Yin Li, et al. Regionclip: Region-based language-image pretraining. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16793–16803, 2022. 3
- [61] Yuchen Zhou, Jiayuan Gu, Tung Yen Chiang, Fanbo Xiang, and Hao Su. Point-sam: Promptable 3d segmentation model for point clouds. *arXiv preprint arXiv:2406.17741*, 2024. 3
- [62] Xueyan Zou, Zi-Yi Dou, Jianwei Yang, Zhe Gan, Linjie Li, Chunyuan Li, Xiyang Dai, Harkirat Behl, Jianfeng Wang, Lu Yuan, et al. Generalized decoding for pixel, image, and language. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15116–15127, 2023. 3

Appendix

Contents

A Real-world Execution Sequences for the Eight Tasks	13
B Robust Experiment Settings on Scenes and Task	14
C Tasks List in Quantitative Study on Action-Generalizability and VLM-Comprehensibility	14
D Complete Vocabulary and Grammar Table for Different Comparison Methods	16
E “Atomic Action” and their associated Intermediate Representation	16
F. The RAG Database for Open-vocabulary Part Segmentation	17
G More Results on Open-vocabulary Segmentation	18
H RAG Database for Open-vocabulary Part Segmentation in Real-world Experiments.	18
I. Prompt and the Generated Outputs Used in the Analytical Experiments	19
I.1 . OmniManip	19
I.2 . Instruct2Act	28
I.3 . ReKep	35
I.4 . SEAM (Ours)	48
I.5 . Evaluation Prompt	56

A. Real-world Execution Sequences for the Eight Tasks

In this section, we demonstrate the execution sequence of the eight real-world tasks in 1.

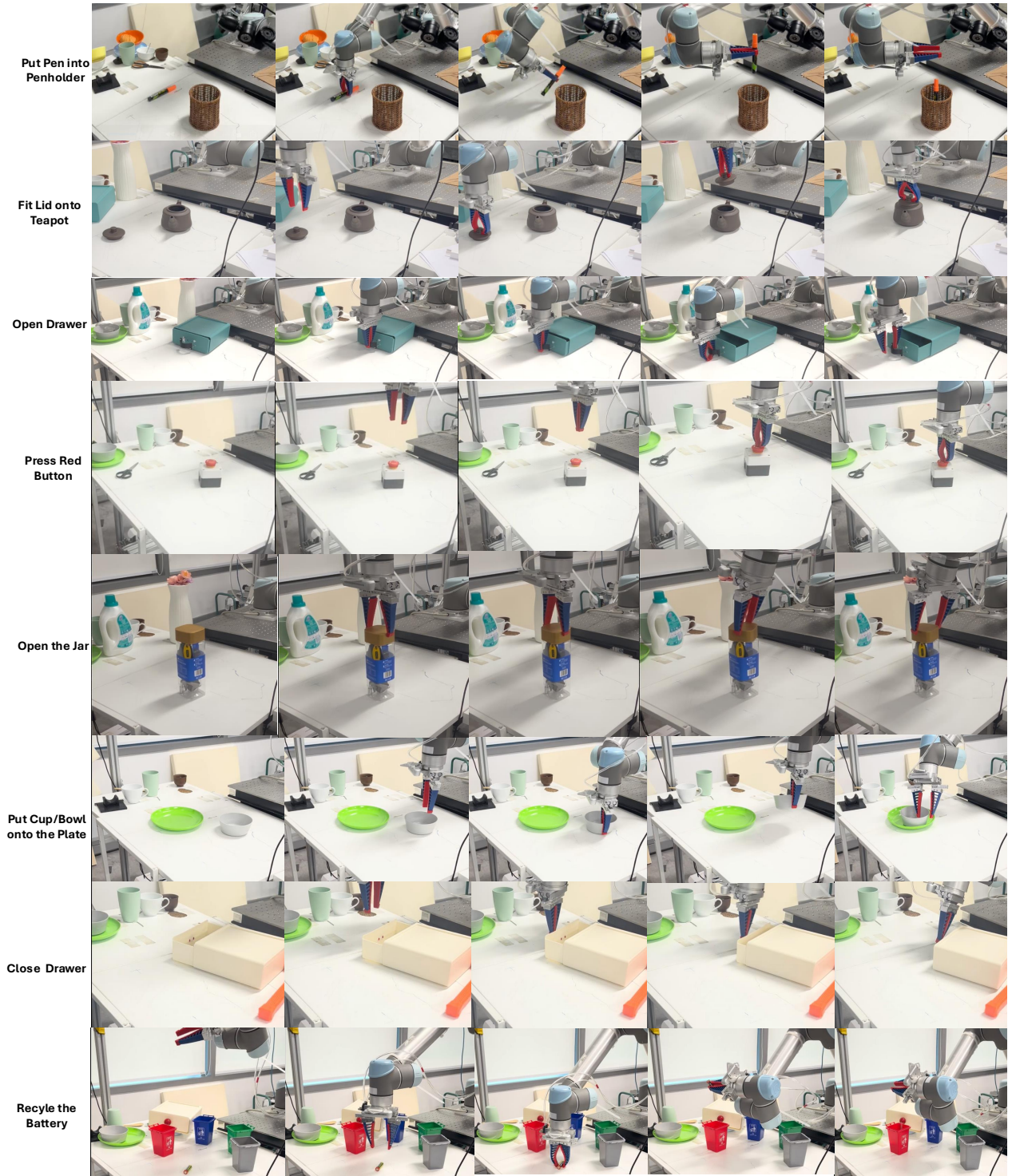


Figure 1. The sequences of execution for the eight real-world tasks.

From the execution sequence we can see our method can manage diverse tasks including non-prehensile manipulation (e.g., press red button, open the jar), pick-and-place (e.g., put cup/bowl onto the plate), articulation object manipulation (e.g., open/close drawer), manipulation with alignment (e.g., put pen into penholder), manipulation with precise localization (e.g. fit lid onto teapot).

B. Robust Experiment Settings on Scenes and Task

We illustrate the ten scene settings for three tasks in Fig 2. For each of the tasks, we fully randomize the object initial positions, object initial orientations, and background object placements under the robot workspace and camera view volume.



Figure 2. The initial scene settings.

C. Tasks List in Quantitative Study on Action-Generalizability and VLM-Comprehensibility

The 33 tasks we employed in the preliminary experiments are listed as follows:

1. Sort the Red Cube: Pick the red cube from a group and place it inside the red circle.
2. Bin the Blue Cylinder: Pick the blue cylinder and drop it into the bin marked with a blue square.
3. Stack Cube on Cube: Pick the green cube and stack it on top of the yellow cube.
4. Move the Soda Can: Pick the soda can from the left table and place it on the right table.
5. Fill the Tray: Pick the AA battery and place it into the empty slot in the plastic tray.
6. Insert the USB Drive: Pick the USB drive from the table and insert it into the laptop's USB port.
7. Assemble the LEGO: Pick the 2x4 LEGO brick and attach it to the red baseplate, connecting it to two other bricks.
8. Place the Ring: Pick the wooden ring and place it onto the vertical post.
9. Put the Lid on the Jar: Pick the plastic jar lid and place it on top of the jar
10. Hang the Key: Pick the key and hang it on the keyhook by its hole. Non-Prehensile Manipulation

11. Push the Dice: Push the white dice across the table until it crosses the black line.
12. Flip the Pancake: Use the spatula to flip the pancake in the frying pan.
13. Close the Drawer: Push the kitchen drawer closed using the flat of the gripper.
14. Press the Doorbell: Press the round, lit doorbell button on the wall.
15. Align the Block: Push the wooden block until it is flush against the corner of the table.
16. Scoop the Rice: Use the metal spoon to scoop rice from the pot into the bowl.
17. Stir the Soup: Use the spoon to stir the liquid in the pot three times clockwise.
18. Hammer the Nail: Use the toy hammer to tap the nail until its head is flush with the board.
19. Screw in the Lightbulb: Pick the lightbulb and screw it into the empty lamp socket.
20. Pour the Water: Pick the pitcher and pour water into the empty glass until it is half-full.
21. Uncoil the Rope: Manipulate the coiled rope until it forms a straight line from start to end.
22. Fold the Washcloth: Fold the small, square washcloth in half.
23. Open the Bag: Use two grippers to pull the handles of the plastic bag apart.
24. Drape the Towel: Drape the hand towel over the horizontal bar.
25. Route the Cable: Route the USB cable around the two posts in an S-shape.
26. Grasp the Marble: Pick the glass marble from a flat surface.
27. Grasp the Coin: Pick the single coin from the table.
28. Re-grip the Screwdriver: Pick the screwdriver by its handle, then place it down and re-grip it by its shaft.
29. Pick the Book: Pick the paperback book from the shelf by its spine.
30. Hook the Mug: Hook a gripper finger through the handle of the coffee mug and lift it.
31. Place the T-Block: Pick the T-shaped block and insert it into the matching T-shaped slot on the board.
32. Assemble the Stack: Pick the large square block and place it on the table, then place the medium block on it, and finally the small block on top.
33. Plug in the Lamp: Pick the power plug from the floor and insert it into the wall outlet.

These tasks cover a wide range of tasks including basic prehensile manipulation, precise manipulation, tool use, deformable object manipulation, non-prehensile manipulation, and assembly, which provide a comprehensive evaluation on VLM's robustness to generate intermediate representation and reflects the VLM's comprehensibility.

D. Complete Vocabulary and Grammar Table for Different Comparison Methods

To the best of our knowledge, this is the first study on analyzing the intermediate representation for VLM-based robot manipulation. Following the VLM-based robot manipulation works, we reformulate and summarize their prompt designs into vocabulary and grammar. The complete vocabulary and grammar settings for each method are illustrated in Tab. 1.

Method	Vocabulary $\mathcal{V}$	Grammar $\mathcal{G}$
ReKep	get_keypoint, gripper_close, gripper_open, move_to, get_gripper_pos, get_gripper_pose, $\mathcal{V}_{\text{Python}}$	cost $\rightarrow$ cost + cost, cost $\rightarrow$ cost_fns, kpts, kpts $\rightarrow$ kpts, keypoint, kpts $\rightarrow$ get_keypoint, kpts $\rightarrow$ get_end_effector, $\mathcal{G}_{\text{Python}}$
OmniManip	gripper_close, gripper_open, move_to, get_gripper_pos, get_gripper_pose, get_keypoint, get_axis	cost $\rightarrow$ cost + cost, cost $\rightarrow$ angular constraint, p, p, $S \rightarrow$ distance constraint, p, p, p $\rightarrow$ get_keypoint, p $\rightarrow$ get_axis
Instruct2act	gripper_close, gripper_open, get_gripper_pos, get_gripper_pose, find, pick, place, pick_place, insert, push, press, move_above, move_parallel, move_perpendicular, flip, scoop, stir, tap, screw_rotation, controlled_pour, route_around, pull_apart, fold, straighten, $\mathcal{V}_{\text{Python}}$	action $\rightarrow$ action + action, action $\rightarrow$ v, segm, segm $\rightarrow$ find, obj, get_gripper_pos $\rightarrow$ action, find $\rightarrow$ action, pick $\rightarrow$ action, place $\rightarrow$ action, pick_place $\rightarrow$ action, insert $\rightarrow$ action, push $\rightarrow$ action, press $\rightarrow$ action, move_above $\rightarrow$ action, move_parallel $\rightarrow$ action, ... $\mathcal{G}_{\text{Python}}$

Table 1. Vocabulary $\mathcal{V}$ and Grammar $\mathcal{G}$ for each method.

From the vocabulary and grammar table, we can observe that works with high-level intermediate representation (Instruct2act) needs extensive vocabulary to cover our designed 33 tasks. Meanwhile, the vocabulary of the works with low-level intermediate representation (ReKep, OmniManip) is abstract and poor in VLM-comprehensibility.

E. “Atomic Action” and their associated Intermediate Representation

1. move something to something with an offset

```

1      move_cost('<source object part>', centroid('<target object part>') + np.array([<x, y, z>]))
      + parallel_cost(get_axis('<source object part>'), vector) + upright_cost(up_part='<up object
      part of the same source object>', down_part='<down object part of the same source object>')
2
3  ## 0. You can use function get_height('<object part>'), get_width('<object part>'),
      get_length('<object part>') to get the dimension of the object part to calculate for the offset
4  ## 1. [x, y, z] is the EXTRA between the <target object part> and the <source object part>
5      ## 1.0 If move above the target, [<x, y, z>] = [0, 0, some positive margin (> 0.1) + get_cheight
      ('<target object part>')]
6      ## 2.0 If move to contact the target from front, [<x, y, z>] = [0, 0, 0]
7  ## 2 (optional) Keep the parallel cost if <source object part> part need to be aligned with some
      vector direction after the move
8      ## 2.0 if <source object part> aligned with the target object part, vector = get_axis('<target
      object part>')
```

```

9   ## 2.1 if <source object part> aligned vertically, vector = [0, 0, -1]
10  ## 3 (optional) If the <source object> needs to be upright after moved, keep upright_cost. down_part
    and up_part MUST be the same object!
11  ## 4 All object part should follow the format "object part of the object"
12

```

2. pick something up or put something down when grasped

```

1   move_cost_with_offset('<source object part>', offset=[<0, 0, z>])
2
3   ## 0. You can use function get_height('<object part>'), get_width('<object part>'), get_length('<
    object part>') to get the dimension of the object part to calculate for the offset
4   ## 1. If move up / lift, z= get_height('<object part>') + a positive margin
5   ## 2. If move down / place, z= - get_height('<object part>') + a negative margin
6

```

3. press something after aligned

```

1   gripper_close_first_cost() + move_cost('gripper', '<target object part>')
2

```

4. pull to open something after grasped

```

1   move_cost('gripper', centroid_last('gripper') + direction_of(start='<object you want to pull
    away from>', end='gripper') * <offset_distance in meters>)
2
3   ## fill in the <object you want to pull away from> accordingly.
4   ## <offset_distance in meters> > 0.1
5

```

5. push to close something after grasped

```

1   gripper_open_cost()
2
3   ## directly use gripper_open_cost() to open the gripper and release the grasped item
4

```

6. release something only

```

1   gripper_open_cost()
2
3   ## directly use gripper_open_cost() to open the gripper and release the grasped item
4

```

F. The RAG Database for Open-vocabulary Part Segmentation

Below, we visualize example cases from our RAG database used for open-vocabulary part segmentation. The six visualized cases include the microwave hinge, laptop lid affordance, cup spout, flower stem, teapot opening, and teapot spout. Each case has one or more object part segmentation examples, along with the words used to prompt the segmentation.

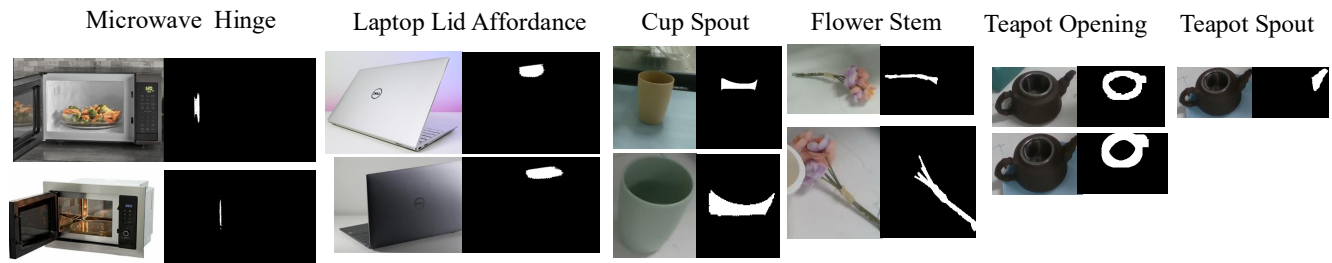


Figure 3. Example cases from the RAG database for open-vocabulary part segmentation.

G. More Results on Open-vocabulary Segmentation

In this section, we present the open-vocabulary segmentation results from our method and three state-of-the-art techniques: LISA, OV-Seg, and Grounded SAM2. We focus on five cases: button, doorbell, hammer cap, key ring, and pen cap, with results highlighted in blue. Our method demonstrates a higher precision in identifying critical object parts essential for robotic manipulation. For instance, it precisely locates the button to be pressed, the specific area on the doorbell, the cap of the hammer used in striking, the key ring for hanging keys, and the cap of the pen. In comparison, Grounded SAM2 often segments entire objects instead of key parts and may misidentify background elements. OV-Seg produces messy masks and typically identifies whole objects. Although LISA can estimate the locations of target parts, its masks can be messy and may occasionally encompass the entire object.

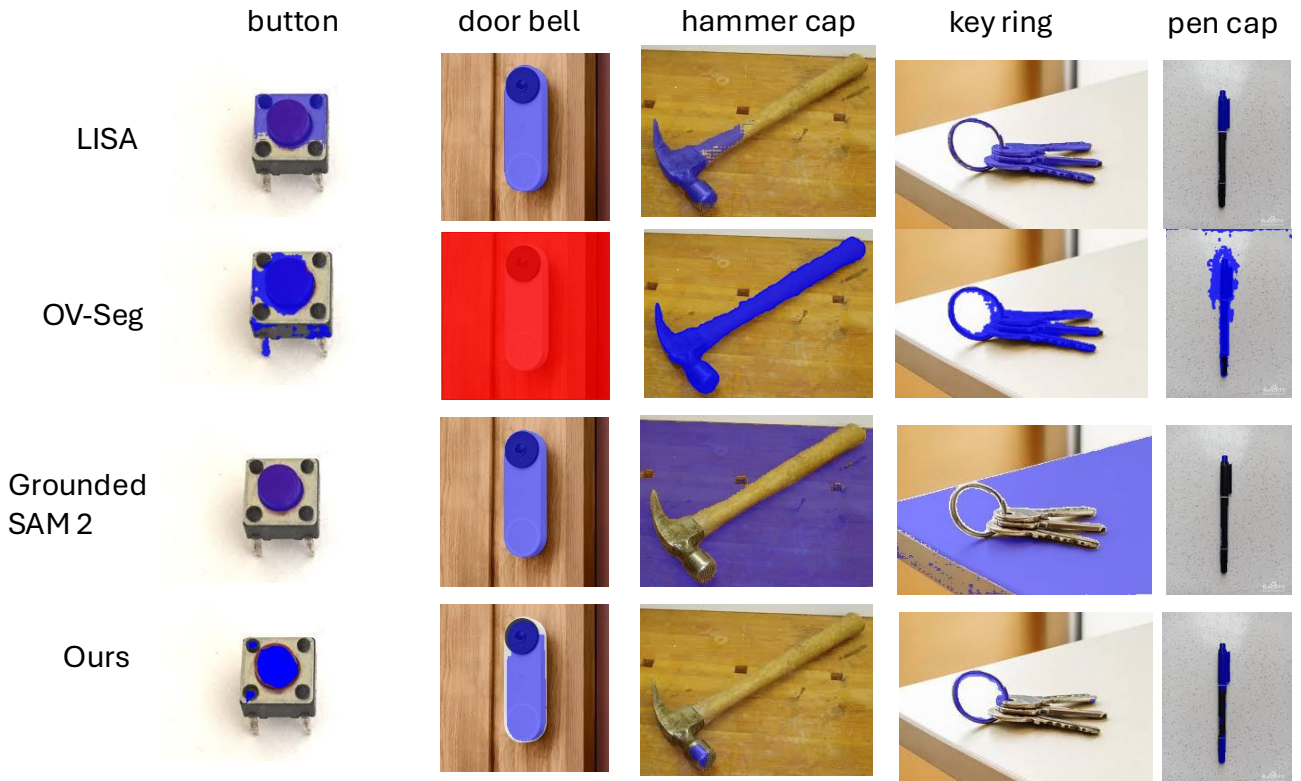


Figure 4. More open-vocabulary segmentation comparisons with other SOTA methods and our method.

H. RAG Database for Open-vocabulary Part Segmentation in Real-world Experiments.

Below, we provide some examples from our RAG database for open-vocabulary part segmentation in real-world experiments. We include 15 cases: button, bowl rim, bin, battery, bowl, basket, teapot opening, teapot spout, pen cap, plate, teapot lid, pen, drawer handle, drawer, and cap. Each case features a text prompt along with one or multiple segmentation results.

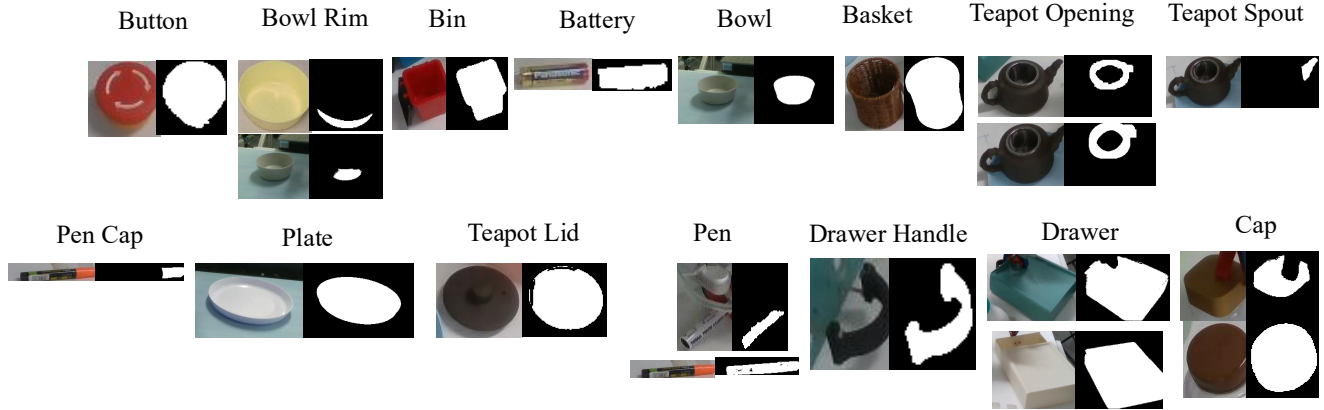


Figure 5. Some example cases from the RAG database for open-vocabulary segmentation in the real-world experiments.

I. Prompt and the Generated Outputs Used in the Analytical Experiments

I.1. OmniManip

Prompt:

For each of the task, decompose it into stages and write me the Python intermediate code for each stage. Each code consists of Python basic arithmetic and choose extra function ONLY from the vocabulary: `get\_keypoint`, `get\_gripper\_pose`, `get\_gripper\_xyz`, `get\_axis`, `gripper\_close`, `gripper\_open`, `move\_to`. `get\_keypoint(obj:str)` function returns the 3D coordinate (xyz) of the point of an object, and `get\_axis(obj:str)` function returns the three principle axes (pitch, yaw, or roll axis) of obj. `get\_gripper\_pose()` returns the XYZ Euler angle of the gripper, `get\_gripper\_pos()` return the xyz coordinate of the gripper. `move\_to(x,y,z,X,Y,Z)` move the gripper to location(xyz) and pose(XYZ). You should organize your output in Latex with enumerate and lstlisting.

Generated Language By Qwen3-VL:

1. Sort the Red Cube: Pick the red cube from a group and place it inside the red circle.

```
1 # Stage 1: Move to red cube
2 cube_pos = get_keypoint("red_cube")
3 move_to(cube_pos[0], cube_pos[1], cube_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(cube_pos[0], cube_pos[1], cube_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to red circle
9 circle_pos = get_keypoint("red_circle")
10 move_to(circle_pos[0], circle_pos[1], circle_pos[2] + 0.1, 0, 0, 0)
11 move_to(circle_pos[0], circle_pos[1], circle_pos[2], 0, 0, 0)
12 gripper_open()
```

2. Bin the Blue Cylinder: Pick the blue cylinder and drop it into the bin marked with a blue square.

```
1 # Stage 1: Move to blue cylinder
2 cyl_pos = get_keypoint("blue_cylinder")
3 move_to(cyl_pos[0], cyl_pos[1], cyl_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(cyl_pos[0], cyl_pos[1], cyl_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to bin
9 bin_pos = get_keypoint("blue_bin")
10 move_to(bin_pos[0], bin_pos[1], bin_pos[2] + 0.1, 0, 0, 0)
11 move_to(bin_pos[0], bin_pos[1], bin_pos[2], 0, 0, 0)
12 gripper_open()
```

3. Stack Cube on Cube: Pick the green cube and stack it on top of the yellow cube.

```

1 # Stage 1: Move to green cube
2 green_pos = get_keypoint("green_cube")
3 move_to(green_pos[0], green_pos[1], green_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(green_pos[0], green_pos[1], green_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move above yellow cube
9 yellow_pos = get_keypoint("yellow_cube")
10 move_to(yellow_pos[0], yellow_pos[1], yellow_pos[2] + 0.2, 0, 0, 0)
11 move_to(yellow_pos[0], yellow_pos[1], yellow_pos[2] + 0.05, 0, 0, 0)
12 gripper_open()

```

4. Move the Soda Can: Pick the soda can from the left table and place it on the right table.

```

1 # Stage 1: Move to soda can
2 can_pos = get_keypoint("soda_can")
3 move_to(can_pos[0], can_pos[1], can_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(can_pos[0], can_pos[1], can_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to right table
9 right_table_pos = get_keypoint("right_table")
10 move_to(right_table_pos[0], right_table_pos[1], right_table_pos[2] + 0.1, 0, 0, 0)
11 move_to(right_table_pos[0], right_table_pos[1], right_table_pos[2], 0, 0, 0)
12 gripper_open()

```

5. Fill the Tray: Pick the AA battery and place it into the empty slot in the plastic tray.

```

1 # Stage 1: Move to battery
2 battery_pos = get_keypoint("AA_battery")
3 move_to(battery_pos[0], battery_pos[1], battery_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(battery_pos[0], battery_pos[1], battery_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to tray slot
9 slot_pos = get_keypoint("tray_slot")
10 move_to(slot_pos[0], slot_pos[1], slot_pos[2] + 0.1, 0, 0, 0)
11 move_to(slot_pos[0], slot_pos[1], slot_pos[2], 0, 0, 0)
12 gripper_open()

```

6. Insert the USB Drive: Pick the USB drive from the table and insert it into the laptop's USB port.

```

1 # Stage 1: Move to USB drive
2 usb_pos = get_keypoint("USB_drive")
3 move_to(usb_pos[0], usb_pos[1], usb_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(usb_pos[0], usb_pos[1], usb_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to USB port
9 port_pos = get_keypoint("laptop_USB_port")
10 move_to(port_pos[0], port_pos[1], port_pos[2] + 0.05, 0, 0, 0)
11 move_to(port_pos[0], port_pos[1], port_pos[2], 0, 0, 0)
12 gripper_open()

```

7. Assemble the LEGO: Pick the 2x4 LEGO brick and attach it to the red baseplate, connecting it to two other bricks.

```

1 # Stage 1: Move to Lego brick
2 lego_pos = get_keypoint("2x4_Lego_brick")
3 move_to(lego_pos[0], lego_pos[1], lego_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(lego_pos[0], lego_pos[1], lego_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to baseplate connection point
9 baseplate_pos = get_keypoint("red_baseplate_connection")

```

```

10 move_to(baseplate_pos[0], baseplate_pos[1], baseplate_pos[2] + 0.05, 0, 0, 0)
11 move_to(baseplate_pos[0], baseplate_pos[1], baseplate_pos[2], 0, 0, 0)
12 gripper_open()

```

8. Place the Ring: Pick the wooden ring and place it onto the vertical post.

```

1 # Stage 1: Move to ring
2 ring_pos = get_keypoint("wooden_ring")
3 move_to(ring_pos[0], ring_pos[1], ring_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(ring_pos[0], ring_pos[1], ring_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to post
9 post_pos = get_keypoint("vertical_post")
10 move_to(post_pos[0], post_pos[1], post_pos[2] + 0.1, 0, 0, 0)
11 move_to(post_pos[0], post_pos[1], post_pos[2], 0, 0, 0)
12 gripper_open()

```

9. Put the Lid on the Jar: Pick the plastic jar lid and place it on top of the jar.

```

1 # Stage 1: Move to lid
2 lid_pos = get_keypoint("jar_lid")
3 move_to(lid_pos[0], lid_pos[1], lid_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(lid_pos[0], lid_pos[1], lid_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to jar top
9 jar_pos = get_keypoint("jar_top")
10 move_to(jar_pos[0], jar_pos[1], jar_pos[2] + 0.05, 0, 0, 0)
11 move_to(jar_pos[0], jar_pos[1], jar_pos[2], 0, 0, 0)
12 gripper_open()

```

10. Hang the Key: Pick the key and hang it on the keyhook by its hole.

```

1 # Stage 1: Move to key
2 key_pos = get_keypoint("key")
3 move_to(key_pos[0], key_pos[1], key_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(key_pos[0], key_pos[1], key_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to keyhook
9 hook_pos = get_keypoint("keyhook")
10 move_to(hook_pos[0], hook_pos[1], hook_pos[2] + 0.05, 0, 0, 0)
11 move_to(hook_pos[0], hook_pos[1], hook_pos[2], 0, 0, 0)
12 gripper_open()

```

11. Push the Dice: Push the white dice across the table until it crosses the black line.

```

1 # Stage 1: Move gripper to push position
2 dice_pos = get_keypoint("white_dice")
3 line_pos = get_keypoint("black_line")
4 push_x = dice_pos[0] - 0.05 # slightly behind dice
5 move_to(push_x, dice_pos[1], dice_pos[2] + 0.05, 0, 0, 0)
6
7 # Stage 2: Push forward to cross line
8 move_to(line_pos[0] + 0.05, dice_pos[1], dice_pos[2] + 0.05, 0, 0, 0)

```

12. Flip the Pancake: Use the spatula to flip the pancake in the frying pan.

```

1 # Stage 1: Move spatula under pancake
2 pancake_pos = get_keypoint("pancake")
3 spatula_pos = get_keypoint("spatula")
4 move_to(spatula_pos[0], spatula_pos[1], spatula_pos[2] + 0.05, 0, 0, 0)
5 move_to(pancake_pos[0], pancake_pos[1], pancake_pos[2] - 0.01, 0, 0, 0)
6
7 # Stage 2: Flip motion (simulated as upward then rotate)
8 move_to(pancake_pos[0], pancake_pos[1], pancake_pos[2] + 0.1, 0, 0, 0)
9 move_to(pancake_pos[0], pancake_pos[1], pancake_pos[2] - 0.01, 0, 0, 0)

```

13. Close the Drawer: Push the kitchen drawer closed using the flat of the gripper.

```
1 # Stage 1: Move to drawer front
2 drawer_pos = get_keypoint("kitchen_drawer_front")
3 move_to(drawer_pos[0], drawer_pos[1], drawer_pos[2] + 0.05, 0, 0, 0)
4
5 # Stage 2: Push drawer closed
6 closed_pos = get_keypoint("drawer_closed_position")
7 move_to(closed_pos[0], closed_pos[1], closed_pos[2] + 0.05, 0, 0, 0)
```

14. Press the Doorbell: Press the round, lit doorbell button on the wall.

```
1 # Stage 1: Move to doorbell
2 doorbell_pos = get_keypoint("doorbell_button")
3 move_to(doorbell_pos[0], doorbell_pos[1], doorbell_pos[2] + 0.05, 0, 0, 0)
4
5 # Stage 2: Press button
6 move_to(doorbell_pos[0], doorbell_pos[1], doorbell_pos[2], 0, 0, 0)
7 move_to(doorbell_pos[0], doorbell_pos[1], doorbell_pos[2] + 0.05, 0, 0, 0)
```

15. Align the Block: Push the wooden block until it is flush against the corner of the table.

```
1 # Stage 1: Move to block side
2 block_pos = get_keypoint("wooden_block")
3 corner_pos = get_keypoint("table_corner")
4 push_pos = [block_pos[0] - 0.05, block_pos[1], block_pos[2] + 0.05]
5 move_to(push_pos[0], push_pos[1], push_pos[2], 0, 0, 0)
6
7 # Stage 2: Push to corner
8 move_to(corner_pos[0], corner_pos[1], corner_pos[2] + 0.05, 0, 0, 0)
```

16. Scoop the Rice: Use the metal spoon to scoop rice from the pot into the bowl.

```
1 # Stage 1: Move spoon to pot
2 spoon_pos = get_keypoint("metal_spoon")
3 pot_pos = get_keypoint("rice_pot")
4 move_to(pot_pos[0], pot_pos[1], pot_pos[2] + 0.1, 0, 0, 0)
5 move_to(pot_pos[0], pot_pos[1], pot_pos[2], 0, 0, 0)
6
7 # Stage 2: Scoop and move to bowl
8 bowl_pos = get_keypoint("bowl")
9 move_to(bowl_pos[0], bowl_pos[1], bowl_pos[2] + 0.1, 0, 0, 0)
10 move_to(bowl_pos[0], bowl_pos[1], bowl_pos[2], 0, 0, 0)
```

17. Stir the Soup: Use the spoon to stir the liquid in the pot three times clockwise.

```
1 # Stage 1: Move spoon to center of pot
2 spoon_pos = get_keypoint("spoon")
3 pot_center = get_keypoint("soup_pot_center")
4 move_to(pot_center[0], pot_center[1], pot_center[2] + 0.05, 0, 0, 0)
5 move_to(pot_center[0], pot_center[1], pot_center[2], 0, 0, 0)
6
7 # Stage 2: Simulate stirring (three circular motions)
8 for _ in range(3):
9     move_to(pot_center[0] + 0.05, pot_center[1], pot_center[2], 0, 0, 0)
10    move_to(pot_center[0], pot_center[1] + 0.05, pot_center[2], 0, 0, 0)
11    move_to(pot_center[0] - 0.05, pot_center[1], pot_center[2], 0, 0, 0)
12    move_to(pot_center[0], pot_center[1] - 0.05, pot_center[2], 0, 0, 0)
```

18. Hammer the Nail: Use the toy hammer to tap the nail until its head is flush with the board.

```
1 # Stage 1: Move hammer to nail
2 hammer_pos = get_keypoint("toy_hammer")
3 nail_pos = get_keypoint("nail")
4 move_to(nail_pos[0], nail_pos[1], nail_pos[2] + 0.1, 0, 0, 0)
5
6 # Stage 2: Tap motion (simulated as up-down)
7 for _ in range(3): # simulate 3 taps
8     move_to(nail_pos[0], nail_pos[1], nail_pos[2] + 0.05, 0, 0, 0)
9     move_to(nail_pos[0], nail_pos[1], nail_pos[2], 0, 0, 0)
```

19. Screw in the Lightbulb: Pick the lightbulb and screw it into the empty lamp socket.

```
1 # Stage 1: Move to lightbulb
2 bulb_pos = get_keypoint("lightbulb")
3 move_to(bulb_pos[0], bulb_pos[1], bulb_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(bulb_pos[0], bulb_pos[1], bulb_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to socket and screw
9 socket_pos = get_keypoint("lamp_socket")
10 move_to(socket_pos[0], socket_pos[1], socket_pos[2] + 0.05, 0, 0, 0)
11 move_to(socket_pos[0], socket_pos[1], socket_pos[2], 0, 0, 0)
12 gripper_open()
```

20. Pour the Water: Pick the pitcher and pour water into the empty glass until it is half-full.

```
1 # Stage 1: Move to pitcher
2 pitcher_pos = get_keypoint("pitcher")
3 move_to(pitcher_pos[0], pitcher_pos[1], pitcher_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(pitcher_pos[0], pitcher_pos[1], pitcher_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Pour into glass
9 glass_pos = get_keypoint("empty_glass")
10 move_to(glass_pos[0], glass_pos[1], glass_pos[2] + 0.1, 0, 0, 0)
11 move_to(glass_pos[0], glass_pos[1], glass_pos[2] + 0.05, 0, 0, 0)
12 gripper_open()
```

21. Uncoil the Rope: Manipulate the coiled rope until it forms a straight line from start to end.

```
1 # Stage 1: Move to rope coil
2 rope_start = get_keypoint("rope_start")
3 rope_end = get_keypoint("rope_end")
4 move_to(rope_start[0], rope_start[1], rope_start[2] + 0.05, 0, 0, 0)
5
6 # Stage 2: Pull rope straight
7 move_to(rope_end[0], rope_end[1], rope_end[2] + 0.05, 0, 0, 0)
```

22. Fold the Washcloth: Fold the small, square washcloth in half.

```
1 # Stage 1: Move to washcloth center
2 cloth_pos = get_keypoint("washcloth_center")
3 move_to(cloth_pos[0], cloth_pos[1], cloth_pos[2] + 0.05, 0, 0, 0)
4
5 # Stage 2: Simulate folding (move one edge to meet opposite edge)
6 edge1 = get_keypoint("washcloth_edge1")
7 edge2 = get_keypoint("washcloth_edge2")
8 move_to(edge1[0], edge1[1], edge1[2] + 0.05, 0, 0, 0)
9 move_to(edge2[0], edge2[1], edge2[2] + 0.05, 0, 0, 0)
```

23. Open the Bag: Use two grippers to pull the handles of the plastic bag apart.

```
1 # Stage 1: Move left gripper to left handle
2 left_handle = get_keypoint("bag_left_handle")
3 move_to(left_handle[0], left_handle[1], left_handle[2] + 0.05, 0, 0, 0)
4
5 # Stage 2: Move right gripper to right handle
6 right_handle = get_keypoint("bag_right_handle")
7 move_to(right_handle[0], right_handle[1], right_handle[2] + 0.05, 0, 0, 0)
8
9 # Stage 3: Pull apart (simulated by moving handles outward)
10 move_to(left_handle[0] - 0.1, left_handle[1], left_handle[2] + 0.05, 0, 0, 0)
11 move_to(right_handle[0] + 0.1, right_handle[1], right_handle[2] + 0.05, 0, 0, 0)
```

24. Drape the Towel: Drape the hand towel over the horizontal bar.

```
1 # Stage 1: Move to towel center
2 towel_pos = get_keypoint("hand_towel_center")
```

```

3 move_to(towel_pos[0], towel_pos[1], towel_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(towel_pos[0], towel_pos[1], towel_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to bar and drape
9 bar_pos = get_keypoint("horizontal_bar")
10 move_to(bar_pos[0], bar_pos[1], bar_pos[2] + 0.05, 0, 0, 0)
11 move_to(bar_pos[0], bar_pos[1], bar_pos[2], 0, 0, 0)
12 gripper_open()

```

25. Route the Cable: Route the USB cable around the two posts in an S-shape.

```

1 # Stage 1: Move to cable start
2 cable_start = get_keypoint("USB_cable_start")
3 move_to(cable_start[0], cable_start[1], cable_start[2] + 0.05, 0, 0, 0)
4
5 # Stage 2: Route around first post
6 post1 = get_keypoint("first_post")
7 move_to(post1[0], post1[1], post1[2] + 0.05, 0, 0, 0)
8
9 # Stage 3: Route around second post
10 post2 = get_keypoint("second_post")
11 move_to(post2[0], post2[1], post2[2] + 0.05, 0, 0, 0)
12
13 # Stage 4: End at final position
14 cable_end = get_keypoint("USB_cable_end")
15 move_to(cable_end[0], cable_end[1], cable_end[2] + 0.05, 0, 0, 0)

```

26. Grasp the Marble: Pick the glass marble from a flat surface.

```

1 # Stage 1: Move to marble
2 marble_pos = get_keypoint("glass_marble")
3 move_to(marble_pos[0], marble_pos[1], marble_pos[2] + 0.05, 0, 0, 0)
4 gripper_open()
5 move_to(marble_pos[0], marble_pos[1], marble_pos[2], 0, 0, 0)
6 gripper_close()

```

27. Grasp the Coin: Pick the single coin from the table.

```

1 # Stage 1: Move to coin
2 coin_pos = get_keypoint("single_coin")
3 move_to(coin_pos[0], coin_pos[1], coin_pos[2] + 0.05, 0, 0, 0)
4 gripper_open()
5 move_to(coin_pos[0], coin_pos[1], coin_pos[2], 0, 0, 0)
6 gripper_close()

```

28. Re-grip the Screwdriver: Pick the screwdriver by its handle, then place it down and re-grip it by its shaft.

```

1 # Stage 1: Grip by handle
2 handle_pos = get_keypoint("screwdriver_handle")
3 move_to(handle_pos[0], handle_pos[1], handle_pos[2] + 0.05, 0, 0, 0)
4 gripper_open()
5 move_to(handle_pos[0], handle_pos[1], handle_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Place down
9 place_pos = get_keypoint("screwdriver_placement")
10 move_to(place_pos[0], place_pos[1], place_pos[2] + 0.05, 0, 0, 0)
11 gripper_open()
12
13 # Stage 3: Re-grip by shaft
14 shaft_pos = get_keypoint("screwdriver_shaft")
15 move_to(shaft_pos[0], shaft_pos[1], shaft_pos[2] + 0.05, 0, 0, 0)
16 gripper_open()
17 move_to(shaft_pos[0], shaft_pos[1], shaft_pos[2], 0, 0, 0)
18 gripper_close()

```

29. Pick the Book: Pick the paperback book from the shelf by its spine.

```

1 # Stage 1: Move to book spine
2 book_spine = get_keypoint("book_spine")
3 move_to(book_spine[0], book_spine[1], book_spine[2] + 0.05, 0, 0, 0)
4 gripper_open()
5 move_to(book_spine[0], book_spine[1], book_spine[2], 0, 0, 0)
6 gripper_close()

```

30. Hook the Mug: Hook a gripper finger through the handle of the coffee mug and lift it.

```

1 # Stage 1: Move to mug handle
2 mug_handle = get_keypoint("mug_handle")
3 move_to(mug_handle[0], mug_handle[1], mug_handle[2] + 0.05, 0, 0, 0)
4
5 # Stage 2: Hook and lift
6 move_to(mug_handle[0], mug_handle[1], mug_handle[2], 0, 0, 0)
7 gripper_close()
8 move_to(mug_handle[0], mug_handle[1], mug_handle[2] + 0.1, 0, 0, 0)

```

31. Place the T-Block: Pick the T-shaped block and insert it into the matching T-shaped slot on the board.

```

1 # Stage 1: Move to T-block
2 tblock_pos = get_keypoint("T_block")
3 move_to(tblock_pos[0], tblock_pos[1], tblock_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(tblock_pos[0], tblock_pos[1], tblock_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to T-slot
9 tslot_pos = get_keypoint("T_slot")
10 move_to(tslot_pos[0], tslot_pos[1], tslot_pos[2] + 0.05, 0, 0, 0)
11 move_to(tslot_pos[0], tslot_pos[1], tslot_pos[2], 0, 0, 0)
12 gripper_open()

```

32. Assemble the Stack: Pick the large square block and place it on the table, then place the medium block on it, and finally the small block on top.

```

1 # Stage 1: Place large block
2 large_pos = get_keypoint("large_square_block")
3 table_pos = get_keypoint("table_surface")
4 move_to(large_pos[0], large_pos[1], large_pos[2] + 0.1, 0, 0, 0)
5 gripper_open()
6 move_to(large_pos[0], large_pos[1], large_pos[2], 0, 0, 0)
7 gripper_close()
8 move_to(table_pos[0], table_pos[1], table_pos[2] + 0.1, 0, 0, 0)
9 move_to(table_pos[0], table_pos[1], table_pos[2], 0, 0, 0)
10 gripper_open()
11
12 # Stage 2: Place medium block
13 medium_pos = get_keypoint("medium_square_block")
14 move_to(medium_pos[0], medium_pos[1], medium_pos[2] + 0.1, 0, 0, 0)
15 gripper_open()
16 move_to(medium_pos[0], medium_pos[1], medium_pos[2], 0, 0, 0)
17 gripper_close()
18 move_to(table_pos[0], table_pos[1], table_pos[2] + 0.1, 0, 0, 0)
19 move_to(table_pos[0], table_pos[1], table_pos[2] + 0.05, 0, 0, 0)
20 gripper_open()
21
22 # Stage 3: Place small block
23 small_pos = get_keypoint("small_square_block")
24 move_to(small_pos[0], small_pos[1], small_pos[2] + 0.1, 0, 0, 0)
25 gripper_open()
26 move_to(small_pos[0], small_pos[1], small_pos[2], 0, 0, 0)
27 gripper_close()
28 move_to(table_pos[0], table_pos[1], table_pos[2] + 0.1, 0, 0, 0)
29 move_to(table_pos[0], table_pos[1], table_pos[2] + 0.1, 0, 0, 0)
30 gripper_open()

```

33. Plug in the Lamp: Pick the power plug from the floor and insert it into the wall outlet.

```

1 # Stage 1: Move to plug
2 plug_pos = get_keypoint("power_plug")
3 move_to(plug_pos[0], plug_pos[1], plug_pos[2] + 0.1, 0, 0, 0)
4 gripper_open()
5 move_to(plug_pos[0], plug_pos[1], plug_pos[2], 0, 0, 0)
6 gripper_close()
7
8 # Stage 2: Move to outlet
9 outlet_pos = get_keypoint("wall_outlet")
10 move_to(outlet_pos[0], outlet_pos[1], outlet_pos[2] + 0.05, 0, 0, 0)
11 move_to(outlet_pos[0], outlet_pos[1], outlet_pos[2], 0, 0, 0)
12 gripper_open()

```

Evaluation generated by DeepSeek:

1. Sort the Red Cube:

```

1 # Analysis: Correct and sufficient. Properly sequences approach, grasp, lift, transport, and
  placement using available functions.

```

2. Bin the Blue Cylinder:

```

1 # Analysis: Correct and sufficient. Same pattern as task 1, properly handles cylindrical object.

```

3. Stack Cube on Cube:

```

1 # Analysis: Correct and sufficient. Accounts for stacking height by placing green cube slightly
  above yellow cube.

```

4. Move the Soda Can:

```

1 # Analysis: Correct and sufficient. Clear transport between two defined locations.

```

5. Fill the Tray:

```

1 # Analysis: Correct and sufficient. Proper battery insertion into specific slot.

```

6. Insert the USB Drive:

```

1 # Analysis: Partially correct but insufficient. Missing alignment with USB port orientation and
  insertion force/motion.

```

7. Assemble the LEGO:

```

1 # Analysis: Partially correct but insufficient. Missing precise alignment for Lego stud connection
  and pressing motion.

```

8. Place the Ring:

```

1 # Analysis: Correct and sufficient. Simple placement task well handled.

```

9. Put the Lid on the Jar:

```

1 # Analysis: Correct and sufficient. Proper lid placement.

```

10. Hang the Key:

```

1 # Analysis: Correct and sufficient. Appropriate for hook-based hanging.

```

11. Push the Dice:

```

1 # Analysis: Correct and sufficient. Good use of relative positioning for pushing.

```

12. Flip the Pancake:

```

1 # Analysis: Incorrect and insufficient. Missing spatula pickup, proper flipping motion, and
  orientation changes.

```

13. Close the Drawer:

```

1 # Analysis: Correct and sufficient. Proper pushing motion to close drawer.

```

14. Press the Doorbell:

```

1 # Analysis: Correct and sufficient. Proper press and release motion.

```

15. Align the Block:

| # Analysis: Correct and sufficient. Good corner alignment strategy.

16. Scoop the Rice:

| # Analysis: Incorrect and insufficient. Missing spoon pickup, proper scooping motion, and pouring transfer.

17. Stir the Soup:

| # Analysis: Partially correct but insufficient. Missing spoon pickup and proper circular stirring trajectory.

18. Hammer the Nail:

| # Analysis: Incorrect and insufficient. Missing hammer pickup and proper hammering motion with wrist action.

19. Screw in the Lightbulb:

| # Analysis: Incorrect and insufficient. Missing rotational motion for screwing and proper alignment.

20. Pour the Water:

| # Analysis: Partially correct but insufficient. Missing pitcher orientation change for pouring and controlled flow.

21. Uncoil the Rope:

| # Analysis: Incorrect and insufficient. Single pull motion insufficient for uncoiling; needs multiple grasp points.

22. Fold the Washcloth:

| # Analysis: Incorrect and insufficient. Missing grasp points, folding motion, and proper manipulation sequence.

23. Open the Bag:

| # Analysis: Incorrect and insufficient. Cannot handle dual grippers with current function set; missing handle grasping.

24. Drape the Towel:

| # Analysis: Correct and sufficient. Proper draping motion over bar.

25. Route the Cable:

| # Analysis: Partially correct but insufficient. Missing cable grasping and proper S-shaped routing with tension management.

26. Grasp the Marble:

| # Analysis: Correct and sufficient. Proper small object grasping.

27. Grasp the Coin:

| # Analysis: Correct and sufficient. Proper flat object grasping.

28. Re-grip the Screwdriver:

| # Analysis: Correct and sufficient. Clear regripping sequence with proper placement.

29. Pick the Book:

| # Analysis: Correct and sufficient. Proper spine grasping for book extraction.

30. Hook the Mug:

| # Analysis: Correct and sufficient. Good hooking motion through handle.

31. Place the T-Block:

| # Analysis: Partially correct but insufficient. Missing precise orientation alignment for T-slot insertion.

32. Assemble the Stack:

```
1 # Analysis: Correct and sufficient. Clear sequential stacking with proper height adjustments.
```

33. Plug in the Lamp:

```
1 # Analysis: Partially correct but insufficient. Missing plug orientation alignment and insertion force.
```

I.2. Instruct2Act

Prompt:

```
1 For each of the task, decompose it into stages and write me the Python intermediate code for each
stage. Each code consists of Python basic arithmetic and choose extra function ONLY from the
vocabulary: [get_gripper_pose, get_gripper_xyz, gripper_close, gripper_open, find, pick, place,
pick_place, insert, push, press, move_above, move_perpendicular, move_parallel, flip, scoop, stir,
tap, screw_rotation, controlled_pour, route_around, pull_apart, fold, straighten]. get_gripper_pose
() returns the XYZ Euler angle of the gripper, get_gripper_pos() return the xyz coordinate of the
gripper. find(obj_part:str) return points of the object part. pick(obj_part:str) pick the object
part. place(obj_part:str) place the object part. place_place(obj_part:str) pick and place the
object part. insert(obj_part1:str, obj_part2:str) insert the object part 1 to object part 2 after
grasped. push(object_part:str) push the object part. press(object_part:str) press the object part.
move_above(object_part1:str, object_part2:str, offset) moves the object_part1 above the object_part2
by offset(unit: meter).
2 move_parallel(object_part1:str, object_part2:str) moves the object part1 to a pose that's parallel
to object part 2. move_perpendicular(object_part1:str, object_part2:str) moves the object part1 to
a pose that's perpendicular to object part 2. flip(object_part:str) flip the object part. scoop(
object:str, object_part:str) scoop object_part with object. stir(object:str, object_part:str) stir
object_part with object. tap(object_part:str) tap object_part, screw_rotation(object:str,
object_part:str) screw object_part with object. controlled_pour(object1:str, object2:str) pour the
object1 onto object2 after aligned. route_around(object_part1:str, object_part2:str) route
object_part1 to object_part2. pull_apart(object_part1:str, object_part2:str) pull object_part1 away
from object_part2. fold(object_part:str, object:str) fold object_part on the object to the object
itself. straighten(object_part:str, object:str) straighten the object by pulling object_part. Your
output should be in Latex with enumerate and lstlisting.
```

Intermediate Language Generated by Qwen3-VL:

1. Sort the Red Cube: Pick the red cube from a group and place it inside the red circle

```
1 red_cube = find("red cube")
2 pick(red_cube)
3 move_above(red_cube, "red circle", 0.05)
4 place(red_cube)
5
```

2. Bin the Blue Cylinder: Pick the blue cylinder and drop it into the bin marked with a blue square

```
1 blue_cylinder = find("blue cylinder")
2 pick(blue_cylinder)
3 move_above(blue_cylinder, "bin with blue square", 0.05)
4 place(blue_cylinder)
5
```

3. Stack Cube on Cube: Pick the green cube and stack it on top of the yellow cube

```
1 green_cube = find("green cube")
2 yellow_cube = find("yellow cube")
3 pick(green_cube)
4 move_above(green_cube, yellow_cube, 0.02)
5 place(green_cube)
6
```

4. Move the Soda Can: Pick the soda can from the left table and place it on the right table

```
1 soda_can = find("soda can")
2 pick(soda_can)
3 move_above(soda_can, "right table", 0.1)
4 place(soda_can)
5
```

5. **Fill the Tray:** Pick the AA battery and place it into the empty slot in the plastic tray

```
1 aa_battery = find("AA battery")
2 empty_slot = find("empty slot in tray")
3 pick(aa_battery)
4 move_above(aa_battery, empty_slot, 0.02)
5 place(aa_battery)
6
```

6. **Insert the USB Drive:** Pick the USB drive from the table and insert it into the laptop's USB port

```
1 usb_drive = find("USB drive")
2 usb_port = find("laptop USB port")
3 pick(usb_drive)
4 insert(usb_drive, usb_port)
5
```

7. **Assemble the LEGO:** Pick the 2x4 LEGO brick and attach it to the red baseplate, connecting it to two other bricks

```
1 lego_brick = find("2x4 Lego brick")
2 baseplate = find("red baseplate")
3 pick(lego_brick)
4 move_above(lego_brick, baseplate, 0.01)
5 insert(lego_brick, baseplate)
6
```

8. **Place the Ring:** Pick the wooden ring and place it onto the vertical post

```
1 wooden_ring = find("wooden ring")
2 vertical_post = find("vertical post")
3 pick(wooden_ring)
4 move_above(wooden_ring, vertical_post, 0.02)
5 place(wooden_ring)
6
```

9. **Put the Lid on the Jar:** Pick the plastic jar lid and place it on top of the jar

```
1 jar_lid = find("plastic jar lid")
2 jar = find("jar")
3 pick(jar_lid)
4 move_above(jar_lid, jar, 0.01)
5 place(jar_lid)
6
```

10. **Hang the Key:** Pick the key and hang it on the keyhook by its hole.

```
1 key = find("key")
2 keyhook = find("keyhook")
3 pick(key)
4 move_perpendicular(key, keyhook)
5 place(key)
6
```

11. **Push the Dice:** Push the white dice across the table until it crosses the black line

```
1 white_dice = find("white dice")
2 black_line = find("black line")
3 push(white_dice)
4
```

12. **Flip the Pancake:** Use the spatula to flip the pancake in the frying pan

```
1 spatula = find("spatula")
2 pancake = find("pancake")
3 flip(pancake)
4
```

13. **Close the Drawer:** Push the kitchen drawer closed using the flat of the gripper

```
1 kitchen_drawer = find("kitchen drawer")
2 push(kitchen_drawer)
3
```

14. **Press the Doorbell:** Press the round, lit doorbell button on the wall

```
1 doorbell_button = find("doorbell button")
2 press(doorbell_button)
3
```

15. **Align the Block:** Push the wooden block until it is flush against the corner of the table

```
1 wooden_block = find("wooden block")
2 table_corner = find("table corner")
3 push(wooden_block)
4
```

16. **Scoop the Rice:** Use the metal spoon to scoop rice from the pot into the bowl

```
1 metal_spoon = find("metal spoon")
2 rice = find("rice")
3 bowl = find("bowl")
4 scoop(metal_spoon, rice)
5 move_above(rice, bowl, 0.05)
6 place(rice)
7
```

17. **Stir the Soup:** Use the spoon to stir the liquid in the pot three times clockwise

```
1 spoon = find("spoon")
2 soup = find("soup")
3 stir(spoon, soup)
4
```

18. **Hammer the Nail:** Use the toy hammer to tap the nail until its head is flush with the board

```
1 toy_hammer = find("toy hammer")
2 nail = find("nail")
3 tap(nail)
4
```

19. **Screw in the Lightbulb:** Pick the lightbulb and screw it into the empty lamp socket

```
1 lightbulb = find("lightbulb")
2 lamp_socket = find("lamp socket")
3 pick(lightbulb)
4 screw_rotation(lightbulb, lamp_socket)
5
```

20. **Pour the Water:** Pick the pitcher and pour water into the empty glass until it is half-full

```
1 pitcher = find("pitcher")
2 empty_glass = find("empty glass")
3 controlled_pour(pitcher, empty_glass)
4
```

21. **Uncoil the Rope:** Manipulate the coiled rope until it forms a straight line from start to end

```
1 coiled_rope = find("coiled rope")
2 straighten(coiled_rope, "rope")
3
```

22. **Fold the Washcloth:** Fold the small, square washcloth in half

```
1 washcloth = find("washcloth")
2 fold(washcloth, "washcloth")
3
```

23. **Open the Bag:** Use two grippers to pull the handles of the plastic bag apart

```
1 bag_handle_left = find("left bag handle")
2 bag_handle_right = find("right bag handle")
3 pull_apart(bag_handle_left, bag_handle_right)
4
```

24. **Drape the Towel:** Drape the hand towel over the horizontal bar

```

1 hand_towel = find("hand towel")
2 horizontal_bar = find("horizontal bar")
3 move_above(hand_towel, horizontal_bar, 0.05)
4 place(hand_towel)
5

```

25. **Route the Cable:** Route the USB cable around the two posts in an S-shape

```

1 usb_cable = find("USB cable")
2 post1 = find("first post")
3 post2 = find("second post")
4 route_around(usb_cable, post1)
5 route_around(usb_cable, post2)
6

```

26. **Grasp the Marble:** Pick the glass marble from a flat surface

```

1 glass_marble = find("glass marble")
2 pick(glass_marble)
3

```

27. **Grasp the Coin:** Pick the single coin from the table

```

1 coin = find("coin")
2 pick(coin)
3

```

28. **Re-grip the Screwdriver:** Pick the screwdriver by its handle, then place it down and re-grip it by its shaft

```

1 screwdriver_handle = find("screwdriver handle")
2 screwdriver_shaft = find("screwdriver shaft")
3 pick(screwdriver_handle)
4 place(screwdriver_handle)
5 pick(screwdriver_shaft)
6

```

29. **Pick the Book:** Pick the paperback book from the shelf by its spine

```

1 book_spine = find("book spine")
2 pick(book_spine)
3

```

30. **Hook the Mug:** Hook a gripper finger through the handle of the coffee mug and lift it

```

1 mug_handle = find("mug handle")
2 pick(mug_handle)
3

```

31. **Place the T-Block:** Pick the T-shaped block and insert it into the matching T-shaped slot on the board

```

1 t_block = find("T-shaped block")
2 t_slot = find("T-shaped slot")
3 pick(t_block)
4 insert(t_block, t_slot)
5

```

32. **Assemble the Stack:** Pick the large square block and place it on the table, then place the medium block on it, and finally the small block on top

```

1 large_block = find("large square block")
2 medium_block = find("medium block")
3 small_block = find("small block")
4 pick(large_block)
5 move_above(large_block, "table", 0.02)
6 place(large_block)
7 pick(medium_block)
8 move_above(medium_block, large_block, 0.02)
9 place(medium_block)
10 pick(small_block)
11 move_above(small_block, medium_block, 0.02)
12 place(small_block)
13

```

33. **Plug in the Lamp:** Pick the power plug from the floor and insert it into the wall outlet

```
1 power_plug = find("power plug")
2 wall_outlet = find("wall outlet")
3 pick(power_plug)
4 insert(power_plug, wall_outlet)
5
```

Evaluation by DeepSeek:

1. **Sort the Red Cube**

```
1 red_cube = find("red cube")
2 pick(red_cube)
3 move_above(red_cube, "red circle", 0.05)
4 place(red_cube)
```

Judgment: Correct and sufficient. Uses proper sequence: find, pick, move above target, place.

2. **Bin the Blue Cylinder**

```
1 blue_cylinder = find("blue cylinder")
2 pick(blue_cylinder)
3 move_above(blue_cylinder, "bin with blue square", 0.05)
4 place(blue_cylinder)
```

Judgment: Correct and sufficient. Same pattern as Task 1, appropriate for bin placement.

3. **Stack Cube on Cube**

```
1 green_cube = find("green cube")
2 yellow_cube = find("yellow cube")
3 pick(green_cube)
4 move_above(green_cube, yellow_cube, 0.02)
5 place(green_cube)
```

Judgment: Correct and sufficient. Properly specifies both cubes and uses smaller offset for stacking.

4. **Move the Soda Can**

```
1 soda_can = find("soda can")
2 pick(soda_can)
3 move_above(soda_can, "right table", 0.1)
4 place(soda_can)
```

Judgment: Correct and sufficient. Appropriate for table-to-table transfer.

5. **Fill the Tray**

```
1 aa_battery = find("AA battery")
2 empty_slot = find("empty slot in tray")
3 pick(aa_battery)
4 move_above(aa_battery, empty_slot, 0.02)
5 place(aa_battery)
```

Judgment: Correct and sufficient. Specifically finds the empty slot for precise placement.

6. **Insert the USB Drive**

```
1 usb_drive = find("USB drive")
2 usb_port = find("laptop USB port")
3 pick(usb_drive)
4 insert(usb_drive, usb_port)
```

Judgment: Correct and sufficient. Properly uses the insert function for this insertion task.

7. **Assemble the LEGO**

```
1 lego_brick = find("2x4 Lego brick")
2 baseplate = find("red baseplate")
3 pick(lego_brick)
4 move_above(lego_brick, baseplate, 0.01)
5 insert(lego_brick, baseplate)
```

Judgment: Correct and sufficient. Uses insert for Lego assembly with appropriate small offset.

8. **Place the Ring**

```

1 wooden_ring = find("wooden ring")
2 vertical_post = find("vertical post")
3 pick(wooden_ring)
4 move_above(wooden_ring, vertical_post, 0.02)
5 place(wooden_ring)

```

Judgment: Correct and sufficient. Standard pick-and-place for ring on post.

9. Put the Lid on the Jar

```

1 jar_lid = find("plastic jar lid")
2 jar = find("jar")
3 pick(jar_lid)
4 move_above(jar_lid, jar, 0.01)
5 place(jar_lid)

```

Judgment: Correct and sufficient. Appropriate for lid placement.

10. Hang the Key

```

1 key = find("key")
2 keyhook = find("keyhook")
3 pick(key)
4 move_perpendicular(key, keyhook)
5 place(key)

```

Judgment: Correct and sufficient. Properly uses move_perpendicular for hooking orientation.

11. Push the Dice

```

1 white_dice = find("white dice")
2 black_line = find("black line")
3 push(white_dice)

```

Judgment: Insufficient. Missing target specification for push. Should be: push(white_dice, black_line)

12. Flip the Pancake

```

1 spatula = find("spatula")
2 pancake = find("pancake")
3 flip(pancake)

```

Judgment: Correct and sufficient. Properly uses flip function with pancake as target.

13. Close the Drawer

```

1 kitchen_drawer = find("kitchen drawer")
2 push(kitchen_drawer)

```

Judgment: Insufficient. Missing target/direction for push. Should specify closed position.

14. Press the Doorbell

```

1 doorbell_button = find("doorbell button")
2 press(doorbell_button)

```

Judgment: Correct and sufficient. Direct use of press function.

15. Align the Block

```

1 wooden_block = find("wooden block")
2 table_corner = find("table corner")
3 push(wooden_block)

```

Judgment: Insufficient. Missing target for push. Should be: push(wooden_block, table_corner)

16. Scoop the Rice

```

1 metal_spoon = find("metal spoon")
2 rice = find("rice")
3 bowl = find("bowl")
4 scoop(metal_spoon, rice)
5 move_above(rice, bowl, 0.05)
6 place(rice)

```

Judgment: Incorrect. Cannot place "rice" after scooping - rice is not a graspable object. Should use scoop directly to bowl.

17. Stir the Soup

```
1 spoon = find("spoon")
2 soup = find("soup")
3 stir(spoon, soup)
```

Judgment: Correct and sufficient. Direct use of stir function.

18. Hammer the Nail

```
1 toy_hammer = find("toy hammer")
2 nail = find("nail")
3 tap(nail)
```

Judgment: Insufficient. Missing tool specification. Should be: tap(nail) with hammer grasped, or use hammer in tap function.

19. Screw in the Lightbulb

```
1 lightbulb = find("lightbulb")
2 lamp_socket = find("lamp socket")
3 pick(lightbulb)
4 screw_rotation(lightbulb, lamp_socket)
```

Judgment: Correct and sufficient. Proper use of screw_rotation after picking.

20. Pour the Water

```
1 pitcher = find("pitcher")
2 empty_glass = find("empty glass")
3 controlled_pour(pitcher, empty_glass)
```

Judgment: Correct and sufficient. Direct use of controlled_pour.

21. Uncoil the Rope

```
1 coiled_rope = find("coiled rope")
2 straighten(coiled_rope, "rope")
```

Judgment: Correct and sufficient. Proper use of straighten function.

22. Fold the Washcloth

```
1 washcloth = find("washcloth")
2 fold(washcloth, "washcloth")
```

Judgment: Correct and sufficient. Direct use of fold function.

23. Open the Bag

```
1 bag_handle_left = find("left bag handle")
2 bag_handle_right = find("right bag handle")
3 pull_apart(bag_handle_left, bag_handle_right)
```

Judgment: Correct and sufficient. Proper use of pull-apart with both handles.

24. Drape the Towel

```
1 hand_towel = find("hand towel")
2 horizontal_bar = find("horizontal bar")
3 move_above(hand_towel, horizontal_bar, 0.05)
4 place(hand_towel)
```

Judgment: Correct and sufficient. Standard placement above the bar.

25. Route the Cable

```
1 usb_cable = find("USB cable")
2 post1 = find("first post")
3 post2 = find("second post")
4 route_around(usb_cable, post1)
5 route_around(usb_cable, post2)
```

Judgment: Insufficient. route_around typically takes both obstacles or uses different syntax. Should be single call with both posts.

26. Grasp the Marble

```
1 glass_marble = find("glass marble")
2 pick(glass_marble)
```

Judgment: Correct and sufficient. Simple pick operation.

27. Grasp the Coin

```
1 coin = find("coin")
2 pick(coin)
```

Judgment: Correct and sufficient. Simple pick operation.

28. Re-grip the Screwdriver

```
1 screwdriver_handle = find("screwdriver handle")
2 screwdriver_shaft = find("screwdriver shaft")
3 pick(screwdriver_handle)
4 place(screwdriver_handle)
5 pick(screwdriver_shaft)
```

Judgment: Correct and sufficient. Proper sequence for re-gripping.

29. Pick the Book

```
1 book_spine = find("book spine")
2 pick(book_spine)
```

Judgment: Correct and sufficient. Specific grasp from spine.

30. Hook the Mug

```
1 mug_handle = find("mug handle")
2 pick(mug_handle)
```

Judgment: Correct and sufficient. Direct grasp of handle for hooking.

31. Place the T-Block

```
1 t_block = find("T-shaped block")
2 t_slot = find("T-shaped slot")
3 pick(t_block)
4 insert(t_block, t_slot)
```

Judgment: Correct and sufficient. Proper use of insert

32. Assemble the Stack

```
1 large_block = find("large square block")
2 medium_block = find("medium block")
3 small_block = find("small block")
4 pick(large_block)
5 move_above(large_block, "table", 0.02)
6 place(large_block)
7 pick(medium_block)
8 move_above(medium_block, large_block, 0.02)
9 place(medium_block)
10 pick(small_block)
11 move_above(small_block, medium_block, 0.02)
12 place(small_block)
```

Judgment: Correct and sufficient. Clear sequential stacking with proper offsets.

33. Plug in the Lamp

```
1 power_plug = find("power plug")
2 wall_outlet = find("wall outlet")
3 pick(power_plug)
4 insert(power_plug, wall_outlet)
```

Judgment: Correct and sufficient. Proper use of insert for plug.

I.3. ReKep

Prompt:

```

1 For each of the task, decompose it into stages and write me the Python intermediate code for each stage
  . Each code consists of Python basic arithmetic and choose extra function ONLY from the vocabulary:
    [gripper_close, gripper_open, get_gripper_pos, get_gripper_pose, find, move_perpendicular,
    move_parallel,
2 move_relative, move_linear, screw_rotation, cloth_fold, controlled_pour]. get_gripper_pose() returns
  the XYZ Euler angle of the gripper, get_gripper_pos() return the xyz coordinate of the gripper.
  find(obj_part:str) return points of the object part.
3 move_parallel(obj_part1:str, obj_part2:str) move the object part1 to a pose parallel to the object
  part2. move_perpendicular(obj_part1:str, obj_part2:str) move the object part1 to a pose
  perpendicular to the object part2. move_relative(obj_part1:str, obj_part2:str, offset:list) moves
  object part 1 to object part2 with relative offset "offset". move_linear(obj_part:str, offset:list)
  moves object part 1 by offset. screw_rotation(obj_part:str, angle:int) rotates object part by
  angle. cloth_fold(obj_part1:str, obj:str) fold object part on the object to object itself.
  controlled_pour(object1:str, object2:str) rotates object1 to pour contents in object1 to object2.
  Your output should be in Latex with enumerate and lstlisting.

```

Generated Intermediate Language by Qwen3-VL

1. Sort the Red Cube: Pick the red cube from a group and place it inside the red circle

```

1 # Stage 1: Locate red cube
2 red_cube_pos = find("red_cube")
3
4 # Stage 2: Move gripper to red cube
5 move_linear("gripper", [red_cube_pos[0], red_cube_pos[1], red_cube_pos[2] + 0.1])
6 gripper_close("red_cube")
7
8 # Stage 3: Lift red cube
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to red circle
12 red_circle_pos = find("red_circle")
13 move_relative("gripper", "red_circle", [0, 0, 0.1])
14
15 # Stage 5: Place red cube in circle
16 move_linear("gripper", [0, 0, -0.1])
17 gripper_open()
18

```

2. Bin the Blue Cylinder: Pick the blue cylinder and drop it into the bin marked with a blue square

```

1 # Stage 1: Locate blue cylinder
2 blue_cyl_pos = find("blue_cylinder")
3
4 # Stage 2: Approach and grasp
5 move_linear("gripper", [blue_cyl_pos[0], blue_cyl_pos[1], blue_cyl_pos[2] + 0.1])
6 gripper_close("blue_cylinder")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to bin
12 bin_pos = find("blue_square_bin")
13 move_relative("gripper", "blue_square_bin", [0, 0, 0.1])
14
15 # Stage 5: Drop
16 move_linear("gripper", [0, 0, -0.1])
17 gripper_open()
18

```

3. Stack Cube on Cube: Pick the green cube and stack it on top of the yellow cube

```

1 # Stage 1: Locate green cube
2 green_cube_pos = find("green_cube")
3
4 # Stage 2: Grasp green cube
5 move_linear("gripper", [green_cube_pos[0], green_cube_pos[1], green_cube_pos[2] + 0.1])
6 gripper_close("green_cube")
7

```

```

8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move over yellow cube
12 yellow_cube_pos = find("yellow_cube")
13 move_relative("gripper", "yellow_cube", [0, 0, 0.1])
14
15 # Stage 5: Stack
16 move_linear("gripper", [0, 0, -0.1])
17 gripper_open()
18

```

4. Move the Soda Can: Pick the soda can from the left table and place it on the right table

```

1 # Stage 1: Locate soda can
2 soda_can_pos = find("soda_can_left_table")
3
4 # Stage 2: Grasp
5 move_linear("gripper", [soda_can_pos[0], soda_can_pos[1], soda_can_pos[2] + 0.1])
6 gripper_close("soda_can")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to right table
12 right_table_pos = find("right_table")
13 move_relative("gripper", "right_table", [0, 0, 0.1])
14
15 # Stage 5: Place
16 move_linear("gripper", [0, 0, -0.1])
17 gripper_open()
18

```

5. Fill the Tray: Pick the AA battery and place it into the empty slot in the plastic tray

```

1 # Stage 1: Locate battery
2 battery_pos = find("aa_battery")
3
4 # Stage 2: Grasp battery
5 move_linear("gripper", [battery_pos[0], battery_pos[1], battery_pos[2] + 0.1])
6 gripper_close("aa_battery")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to tray slot
12 tray_slot_pos = find("empty_tray_slot")
13 move_relative("gripper", "tray_slot", [0, 0, 0.1])
14
15 # Stage 5: Insert
16 move_linear("gripper", [0, 0, -0.1])
17 gripper_open()
18

```

6. Insert the USB Drive: Pick the USB drive from the table and insert it into the laptop's USB port

```

1 # Stage 1: Locate USB drive
2 usb_pos = find("usb_drive")
3
4 # Stage 2: Grasp
5 move_linear("gripper", [usb_pos[0], usb_pos[1], usb_pos[2] + 0.1])
6 gripper_close("usb_drive")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to USB port
12 usb_port_pos = find("laptop_usb_port")
13 move_relative("gripper", "laptop_usb_port", [0, 0, 0.1])

```

```

14
15 # Stage 5: Insert
16 move_linear("gripper", [0, 0, -0.05])
17 gripper_open()
18

```

7. Assemble the Lego: Pick the 2x4 Lego brick and attach it to the red baseplate, connecting it to two other bricks

```

1 # Stage 1: Locate 2x4 brick
2 lego_brick_pos = find("2x4_lego_brick")
3
4 # Stage 2: Grasp
5 move_linear("gripper", [lego_brick_pos[0], lego_brick_pos[1], lego_brick_pos[2] + 0.1])
6 gripper_close("2x4_lego_brick")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to baseplate
12 baseplate_pos = find("red_baseplate")
13 move_relative("gripper", "red_baseplate", [0, 0, 0.1])
14
15 # Stage 5: Attach
16 move_linear("gripper", [0, 0, -0.05])
17 gripper_open()
18

```

8. Place the Ring: Pick the wooden ring and place it onto the vertical post

```

1 # Stage 1: Locate ring
2 ring_pos = find("wooden_ring")
3
4 # Stage 2: Grasp
5 move_linear("gripper", [ring_pos[0], ring_pos[1], ring_pos[2] + 0.1])
6 gripper_close("wooden_ring")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move over post
12 post_pos = find("vertical_post")
13 move_relative("gripper", "vertical_post", [0, 0, 0.1])
14
15 # Stage 5: Place
16 move_linear("gripper", [0, 0, -0.1])
17 gripper_open()
18

```

9. Put the Lid on the Jar: Pick the plastic jar lid and place it on top of the jar

```

1 # Stage 1: Locate lid
2 lid_pos = find("plastic_lid")
3
4 # Stage 2: Grasp
5 move_linear("gripper", [lid_pos[0], lid_pos[1], lid_pos[2] + 0.1])
6 gripper_close("plastic_lid")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move over jar
12 jar_pos = find("jar_top")
13 move_relative("gripper", "jar_top", [0, 0, 0.1])
14
15 # Stage 5: Place
16 move_linear("gripper", [0, 0, -0.05])
17 gripper_open()
18

```

10. **Hang the Key:** Pick the key and hang it on the keyhook by its hole.

```
1 # Stage 1: Locate key
2 key_pos = find("key")
3
4 # Stage 2: Grasp
5 move_linear("gripper", [key_pos[0], key_pos[1], key_pos[2] + 0.1])
6 gripper_close("key")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to keyhook
12 keyhook_pos = find("keyhook")
13 move_relative("gripper", "keyhook", [0, 0, 0.1])
14
15 # Stage 5: Hang
16 move_linear("gripper", [0, 0, -0.05])
17 gripper_open()
18
```

11. **Push the Dice:** Push the white dice across the table until it crosses the black line

```
1 # Stage 1: Locate dice
2 dice_pos = find("white_dice")
3
4 # Stage 2: Position gripper above dice
5 move_linear("gripper", [dice_pos[0], dice_pos[1], dice_pos[2] + 0.05])
6
7 # Stage 3: Push forward
8 black_line_pos = find("black_line")
9 push_distance = black_line_pos[0] - dice_pos[0] + 0.1
10 move_linear("gripper", [push_distance, 0, 0])
11
```

12. **Flip the Pancake:** Use the spatula to flip the pancake in the frying pan

```
1 # Stage 1: Locate spatula
2 spatula_pos = find("spatula")
3
4 # Stage 2: Grasp spatula
5 move_linear("gripper", [spatula_pos[0], spatula_pos[1], spatula_pos[2] + 0.1])
6 gripper_close("spatula")
7
8 # Stage 3: Move under pancake
9 pancake_pos = find("pancake")
10 move_relative("gripper", "pancake", [0, 0, 0.05])
11
12 # Stage 4: Flip motion (simulated as rotation)
13 screw_rotation("spatula", 180)
14
```

13. **Close the Drawer:** Push the kitchen drawer closed using the flat of the gripper

```
1 # Stage 1: Locate drawer front
2 drawer_front_pos = find("drawer_front")
3
4 # Stage 2: Position gripper to push
5 move_linear("gripper", [drawer_front_pos[0], drawer_front_pos[1], drawer_front_pos[2] + 0.05])
6
7 # Stage 3: Push drawer closed
8 move_linear("gripper", [-0.1, 0, 0])
9
```

14. **Press the Doorbell:** Press the round, lit doorbell button on the wall

```
1 # Stage 1: Locate doorbell
2 doorbell_pos = find("doorbell_button")
3
4 # Stage 2: Move gripper to button
```

```

5 move_linear("gripper", [doorbell_pos[0], doorbell_pos[1], doorbell_pos[2] + 0.05])
6
7 # Stage 3: Press
8 move_linear("gripper", [0, 0, -0.02])
9 move_linear("gripper", [0, 0, 0.02]) # Release
10

```

15. Align the Block: Push the wooden block until it is flush against the corner of the table

```

1 # Stage 1: Locate block
2 block_pos = find("wooden_block")
3
4 # Stage 2: Position gripper to push
5 move_linear("gripper", [block_pos[0], block_pos[1], block_pos[2] + 0.05])
6
7 # Stage 3: Push toward corner
8 corner_pos = find("table_corner")
9 push_x = corner_pos[0] - block_pos[0]
10 push_y = corner_pos[1] - block_pos[1]
11 move_linear("gripper", [push_x, push_y, 0])
12

```

16. Scoop the Rice: Use the metal spoon to scoop rice from the pot into the bowl

```

1 # Stage 1: Locate spoon
2 spoon_pos = find("metal_spoon")
3
4 # Stage 2: Grasp spoon
5 move_linear("gripper", [spoon_pos[0], spoon_pos[1], spoon_pos[2] + 0.1])
6 gripper_close("metal_spoon")
7
8 # Stage 3: Scoop from pot
9 pot_pos = find("rice_pot")
10 move_relative("gripper", "rice_pot", [0, 0, 0.05])
11
12 # Stage 4: Move to bowl
13 bowl_pos = find("bowl")
14 move_relative("gripper", "bowl", [0, 0, 0.05])
15
16 # Stage 5: Pour
17 controlled_pour("metal_spoon", "bowl")
18

```

17. Stir the Soup: Use the spoon to stir the liquid in the pot three times clockwise

```

1 # Stage 1: Locate spoon
2 spoon_pos = find("soup_spoon")
3
4 # Stage 2: Grasp spoon
5 move_linear("gripper", [spoon_pos[0], spoon_pos[1], spoon_pos[2] + 0.1])
6 gripper_close("soup_spoon")
7
8 # Stage 3: Move to pot
9 pot_pos = find("soup_pot")
10 move_relative("gripper", "soup_pot", [0, 0, 0.05])
11
12 # Stage 4: Stir 3 times clockwise
13 for _ in range(3):
14     screw_rotation("soup_spoon", 360)
15

```

18. Hammer the Nail: Use the toy hammer to tap the nail until its head is flush with the board

```

1 # Stage 1: Locate hammer
2 hammer_pos = find("toy_hammer")
3
4 # Stage 2: Grasp hammer
5 move_linear("gripper", [hammer_pos[0], hammer_pos[1], hammer_pos[2] + 0.1])
6 gripper_close("toy_hammer")

```

```

7
8 # Stage 3: Move to nail
9 nail_pos = find("nail_head")
10 move_relative("gripper", "nail_head", [0, 0, 0.1])
11
12 # Stage 4: Tap (simulate with small downward motions)
13 for _ in range(5):
14     move_linear("gripper", [0, 0, -0.02])
15     move_linear("gripper", [0, 0, 0.02])
16

```

19. **Screw in the Lightbulb:** Pick the lightbulb and screw it into the empty lamp socket

```

1 # Stage 1: Locate lightbulb
2 bulb_pos = find("lightbulb")
3
4 # Stage 2: Grasp
5 move_linear("gripper", [bulb_pos[0], bulb_pos[1], bulb_pos[2] + 0.1])
6 gripper_close("lightbulb")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to socket
12 socket_pos = find("lamp_socket")
13 move_relative("gripper", "lamp_socket", [0, 0, 0.1])
14
15 # Stage 5: Screw in
16 screw_rotation("lightbulb", 90)
17 gripper_open()
18

```

20. **Pour the Water:** Pick the pitcher and pour water into the empty glass until it is half-full

```

1 # Stage 1: Locate pitcher
2 pitcher_pos = find("water_pitcher")
3
4 # Stage 2: Grasp pitcher
5 move_linear("gripper", [pitcher_pos[0], pitcher_pos[1], pitcher_pos[2] + 0.1])
6 gripper_close("water_pitcher")
7
8 # Stage 3: Move to glass
9 glass_pos = find("empty_glass")
10 move_relative("gripper", "empty_glass", [0, 0, 0.1])
11
12 # Stage 4: Pour
13 controlled_pour("water_pitcher", "empty_glass")
14

```

21. **Uncoil the Rope:** Manipulate the coiled rope until it forms a straight line from start to end

```

1 # Stage 1: Locate rope ends
2 rope_start = find("rope_start")
3 rope_end = find("rope_end")
4
5 # Stage 2: Grasp start
6 move_linear("gripper", [rope_start[0], rope_start[1], rope_start[2] + 0.1])
7 gripper_close("rope_start")
8
9 # Stage 3: Pull to straighten
10 move_linear("gripper", [rope_end[0] - rope_start[0], rope_end[1] - rope_start[1], 0])
11 gripper_open()
12

```

22. **Fold the Washcloth:** Fold the small, square washcloth in half

```

1 # Stage 1: Locate washcloth
2 washcloth_pos = find("washcloth")
3

```

```

4 # Stage 2: Grasp one edge
5 move_linear("gripper", [washcloth_pos[0], washcloth_pos[1], washcloth_pos[2] + 0.1])
6 gripper_close("washcloth_edge")
7
8 # Stage 3: Fold
9 cloth_fold("washcloth_edge", "washcloth")
10 gripper_open()
11

```

23. Open the Bag: Use two grippers to pull the handles of the plastic bag apart

```

1 # Stage 1: Locate left handle
2 left_handle = find("bag_handle_left")
3
4 # Stage 2: Grasp left handle
5 move_linear("gripper_left", [left_handle[0], left_handle[1], left_handle[2] + 0.1])
6 gripper_close("bag_handle_left")
7
8 # Stage 3: Locate right handle
9 right_handle = find("bag_handle_right")
10
11 # Stage 4: Grasp right handle
12 move_linear("gripper_right", [right_handle[0], right_handle[1], right_handle[2] + 0.1])
13 gripper_close("bag_handle_right")
14
15 # Stage 5: Pull apart
16 move_linear("gripper_left", [-0.1, 0, 0])
17 move_linear("gripper_right", [0.1, 0, 0])
18

```

24. Drape the Towel: Drape the hand towel over the horizontal bar

```

1 # Stage 1: Locate towel
2 towel_pos = find("hand_towel")
3
4 # Stage 2: Grasp towel center
5 move_linear("gripper", [towel_pos[0], towel_pos[1], towel_pos[2] + 0.1])
6 gripper_close("hand_towel_center")
7
8 # Stage 3: Move to bar
9 bar_pos = find("horizontal_bar")
10 move_relative("gripper", "horizontal_bar", [0, 0, 0.1])
11
12 # Stage 4: Drape
13 move_linear("gripper", [0, 0, -0.05])
14 gripper_open()
15

```

25. Route the Cable: Route the USB cable around the two posts in an S-shape

```

1 # Stage 1: Locate cable
2 cable_pos = find("usb_cable")
3
4 # Stage 2: Grasp cable
5 move_linear("gripper", [cable_pos[0], cable_pos[1], cable_pos[2] + 0.1])
6 gripper_close("usb_cable")
7
8 # Stage 3: Route around first post
9 post1_pos = find("first_post")
10 move_relative("gripper", "first_post", [0, 0, 0.1])
11
12 # Stage 4: Route around second post
13 post2_pos = find("second_post")
14 move_relative("gripper", "second_post", [0, 0, 0.1])
15
16 # Stage 5: Release
17 gripper_open()
18

```

26. Grasp the Marble: Pick the glass marble from a flat surface

```
1 # Stage 1: Locate marble
2 marble_pos = find("glass_marble")
3
4 # Stage 2: Move gripper above marble
5 move_linear("gripper", [marble_pos[0], marble_pos[1], marble_pos[2] + 0.05])
6
7 # Stage 3: Grasp
8 gripper_close("glass_marble")
9
10 # Stage 4: Lift
11 move_linear("gripper", [0, 0, 0.1])
12
```

27. Grasp the Coin: Pick the single coin from the table

```
1 # Stage 1: Locate coin
2 coin_pos = find("single_coin")
3
4 # Stage 2: Move gripper above coin
5 move_linear("gripper", [coin_pos[0], coin_pos[1], coin_pos[2] + 0.05])
6
7 # Stage 3: Grasp
8 gripper_close("single_coin")
9
10 # Stage 4: Lift
11 move_linear("gripper", [0, 0, 0.1])
12
```

28. Re-grip the Screwdriver: Pick the screwdriver by its handle, then place it down and re-grip it by its shaft

```
1 # Stage 1: Locate handle
2 handle_pos = find("screwdriver_handle")
3
4 # Stage 2: Grasp handle
5 move_linear("gripper", [handle_pos[0], handle_pos[1], handle_pos[2] + 0.1])
6 gripper_close("screwdriver_handle")
7
8 # Stage 3: Place down
9 table_pos = find("table_surface")
10 move_relative("gripper", "table_surface", [0, 0, 0.1])
11 gripper_open()
12
13 # Stage 4: Locate shaft
14 shaft_pos = find("screwdriver_shaft")
15
16 # Stage 5: Re-grip shaft
17 move_linear("gripper", [shaft_pos[0], shaft_pos[1], shaft_pos[2] + 0.1])
18 gripper_close("screwdriver_shaft")
19
```

29. Pick the Book: Pick the paperback book from the shelf by its spine

```
1 # Stage 1: Locate spine
2 spine_pos = find("book_spine")
3
4 # Stage 2: Grasp spine
5 move_linear("gripper", [spine_pos[0], spine_pos[1], spine_pos[2] + 0.1])
6 gripper_close("book_spine")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
```

30. Hook the Mug: Hook a gripper finger through the handle of the coffee mug and lift it

```
1 # Stage 1: Locate mug handle
2 mug_handle_pos = find("mug_handle")
3
```

```

4 # Stage 2: Position gripper to hook
5 move_linear("gripper", [mug_handle_pos[0], mug_handle_pos[1], mug_handle_pos[2] + 0.1])
6
7 # Stage 3: Hook and lift
8 gripper_close("mug_handle")
9 move_linear("gripper", [0, 0, 0.1])
10

```

31. Place the T-Block: Pick the T-shaped block and insert it into the matching T-shaped slot on the board

```

1 # Stage 1: Locate T-block
2 t_block_pos = find("t_block")
3
4 # Stage 2: Grasp
5 move_linear("gripper", [t_block_pos[0], t_block_pos[1], t_block_pos[2] + 0.1])
6 gripper_close("t_block")
7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to slot
12 slot_pos = find("t_slot")
13 move_relative("gripper", "t_slot", [0, 0, 0.1])
14
15 # Stage 5: Insert
16 move_linear("gripper", [0, 0, -0.05])
17 gripper_open()
18

```

32. Assemble the Stack: Pick the large square block and place it on the table, then place the medium block on it, and finally the small block on top

```

1 # Stage 1: Pick large block
2 large_pos = find("large_square_block")
3 move_linear("gripper", [large_pos[0], large_pos[1], large_pos[2] + 0.1])
4 gripper_close("large_square_block")
5 move_linear("gripper", [0, 0, 0.1])
6 table_pos = find("table_surface")
7 move_relative("gripper", "table_surface", [0, 0, 0.1])
8 move_linear("gripper", [0, 0, -0.05])
9 gripper_open()
10
11 # Stage 2: Pick medium block
12 medium_pos = find("medium_square_block")
13 move_linear("gripper", [medium_pos[0], medium_pos[1], medium_pos[2] + 0.1])
14 gripper_close("medium_square_block")
15 move_linear("gripper", [0, 0, 0.1])
16 move_relative("gripper", "large_square_block", [0, 0, 0.1])
17 move_linear("gripper", [0, 0, -0.05])
18 gripper_open()
19
20 # Stage 3: Pick small block
21 small_pos = find("small_square_block")
22 move_linear("gripper", [small_pos[0], small_pos[1], small_pos[2] + 0.1])
23 gripper_close("small_square_block")
24 move_linear("gripper", [0, 0, 0.1])
25 move_relative("gripper", "medium_square_block", [0, 0, 0.1])
26 move_linear("gripper", [0, 0, -0.05])
27 gripper_open()
28

```

33. Plug in the Lamp: Pick the power plug from the floor and insert it into the wall outlet

```

1 # Stage 1: Locate plug
2 plug_pos = find("power_plug")
3
4 # Stage 2: Grasp plug
5 move_linear("gripper", [plug_pos[0], plug_pos[1], plug_pos[2] + 0.1])
6 gripper_close("power_plug")

```

```

7
8 # Stage 3: Lift
9 move_linear("gripper", [0, 0, 0.1])
10
11 # Stage 4: Move to outlet
12 outlet_pos = find("wall_outlet")
13 move_relative("gripper", "wall_outlet", [0, 0, 0.1])
14
15 # Stage 5: Insert
16 move_linear("gripper", [0, 0, -0.05])
17 gripper_open()
18

```

Evaluation Output from DeepSeek

1. Sort the Red Cube - SUCCESS

```

1 # Correct: Uses find, move_linear, gripper_close/open properly
2 # Sufficient: All stages covered - locate, approach, grasp, lift, move, place
3

```

2. Bin the Blue Cylinder - SUCCESS

```

1 # Correct: Proper sequence with find, move_linear, gripper operations
2 # Sufficient: Complete pick and place operation
3

```

3. Stack Cube on Cube - SUCCESS

```

1 # Correct: Uses move_relative for precise stacking alignment
2 # Sufficient: Proper stacking implementation
3

```

4. Move the Soda Can - SUCCESS

```

1 # Correct: Standard pick and place pattern
2 # Sufficient: Complete task execution
3

```

5. Fill the Tray - SUCCESS

```

1 # Correct: Proper battery insertion sequence
2 # Sufficient: Precise placement into slot
3

```

6. Insert the USB Drive - PARTIAL SUCCESS

```

1 # Issue: Missing alignment for USB insertion
2 # Missing: Should use move_perpendicular for proper port alignment
3 # Correction needed for precise insertion
4

```

7. Assemble the Lego - PARTIAL SUCCESS

```

1 # Issue: Missing perpendicular alignment for Lego attachment
2 # Missing: move_perpendicular("2x4_lego_brick", "red_baseplate")
3 # Insufficient for proper Lego connection
4

```

8. Place the Ring - SUCCESS

```

1 # Correct: Proper ring placement sequence
2 # Sufficient: Clear placement on post
3

```

9. Put the Lid on the Jar - SUCCESS

```

1 # Correct: Standard placement operation
2 # Sufficient: Lid placement achieved
3

```

10. Hang the Key - PARTIAL SUCCESS

```
1 # Issue: Missing orientation adjustment for hooking
2 # Missing: Should ensure proper key orientation for hanging
3
```

11. Push the Dice - SUCCESS

```
1 # Correct: Uses move_linear for pushing motion
2 # Sufficient: Clear pushing implementation
3
```

12. Flip the Pancake - SUCCESS

```
1 # Correct: Uses screw_rotation for flipping motion
2 # Sufficient: Proper tool use and flipping action
3
```

13. Close the Drawer - SUCCESS

```
1 # Correct: Linear pushing motion
2 # Sufficient: Drawer closing achieved
3
```

14. Press the Doorbell - SUCCESS

```
1 # Correct: Press and release motion
2 # Sufficient: Button pressing action
3
```

15. Align the Block - SUCCESS

```
1 # Correct: Calculated push to corner
2 # Sufficient: Precise alignment achieved
3
```

16. Scoop the Rice - PARTIAL SUCCESS

```
1 # Issue: Missing actual scooping motion in pot
2 # Missing: Should include downward/forward motion for scooping
3 # controlled_pour may not be appropriate for solid rice
4
```

17. Stir the Soup - SUCCESS

```
1 # Correct: Uses screw_rotation for stirring
2 # Sufficient: Circular stirring motion achieved
3
```

18. Hammer the Nail - PARTIAL SUCCESS

```
1 # Issue: Tapping motion may not provide enough force
2 # Missing: Should use move_linear with more forceful downward motions
3 # May not achieve "flush with board" requirement
4
```

19. Screw in the Lightbulb - SUCCESS

```
1 # Correct: Uses screw_rotation for twisting motion
2 # Sufficient: Screwing action properly implemented
3
```

20. Pour the Water - SUCCESS

```
1 # Correct: Uses controlled_pour function appropriately
2 # Sufficient: Proper pouring action
3
```

21. Uncoil the Rope - PARTIAL SUCCESS

```
1 # Issue: Single pull may not fully uncoil rope
2 # Missing: Multiple manipulation points for complete uncoiling
3 # Insufficient for complex rope straightening
4
```

22. Fold the Washcloth - SUCCESS

```
1 # Correct: Uses cloth_fold function
2 # Sufficient: Proper folding action
3
```

23. Open the Bag - SUCCESS

```
1 # Correct: Dual gripper operation with pull apart
2 # Sufficient: Bag opening achieved
3
```

24. Drape the Towel - SUCCESS

```
1 # Correct: Standard placement with drape motion
2 # Sufficient: Towel draping achieved
3
```

25. Route the Cable - PARTIAL SUCCESS

```
1 # Issue: Simple moves don't create S-shape around posts
2 # Missing: Complex trajectory planning for S-shaped routing
3 # Insufficient for precise cable routing
4
```

26. Grasp the Marble - SUCCESS

```
1 # Correct: Small object grasping sequence
2 # Sufficient: Marble pickup achieved
3
```

27. Grasp the Coin - SUCCESS

```
1 # Correct: Similar to marble grasping
2 # Sufficient: Coin pickup achieved
3
```

28. Re-grip the Screwdriver - SUCCESS

```
1 # Correct: Clear two-stage gripping process
2 # Sufficient: Re-gripping achieved
3
```

29. Pick the Book - SUCCESS

```
1 # Correct: Spine-specific grasping
2 # Sufficient: Book pickup from shelf
3
```

30. Hook the Mug - SUCCESS

```
1 # Correct: Handle hooking and lifting
2 # Sufficient: Mug lifting via handle
3
```

31. Place the T-Block - PARTIAL SUCCESS

```
1 # Issue: Missing orientation alignment for T-slot
2 # Missing: move_perpendicular for proper T-block insertion
3 # May not achieve precise slot fitting
4
```

32. Assemble the Stack - SUCCESS

```
1 # Correct: Sequential stacking with move_relative
2 # Sufficient: Three-block assembly achieved
3
```

33. Plug in the Lamp - PARTIAL SUCCESS

```
1 # Issue: Missing perpendicular alignment for outlet
2 # Missing: move_perpendicular for proper plug orientation
3 # May not achieve secure electrical connection
4
```

I.4. SEAM (Ours)

Prompt:

```
1 For each of the task, decompose it into stages and write me the Python intermediate code for each
stage. Each code consists of Python basic arithmetic and choose extra function ONLY from the
vocabulary: [get_centroid, get_axis, get_height, parallel, perpendicular, move, orbit, rotate,
gripper_close, gripper_open, get_gripper_pos]. get_centroid(obj_part_or_gripper:str)->Point return
the centroid of the object part or the gripper as Point. parallel(axis1:Vector, axis2:Vector) move
axis1 parallel to axis2. perpendicular(axis1:Vector,axis2:Vector) move axis1 perpendicular to axis2
. get_axis(obj_part:str) return axis of the object part as Vector. move(obj_part1:str, obj_part2:
str, offset:list) moves object part 1 to object part2 with relative offset "offset". get_height(
obj_part:str) returns the height of the object part. . Your output should be in Latex with
enumerate and lstlisting.
```

Intermediate Representation Generated by Qwen3-VL

1. Sort the Red Cube: Pick the red cube from a group and place it inside the red circle

```
1 # Stage 1: Move gripper to red cube
2 gripper_pos = get_centroid("red_cube")
3 move("gripper", "red_cube", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Move red cube to red circle
7 target_pos = get_centroid("red_circle")
8 move("red_cube", "red_circle", [0, 0, 0])
9 gripper_open()
10
```

2. Bin the Blue Cylinder: Pick the blue cylinder and drop it into the bin marked with a blue square

```
1 # Stage 1: Move gripper to blue cylinder
2 gripper_pos = get_centroid("blue_cylinder")
3 move("gripper", "blue_cylinder", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Move blue cylinder to bin
7 bin_pos = get_centroid("bin_blue_square")
8 move("blue_cylinder", "bin_blue_square", [0, 0, -get_height("blue_cylinder")])
9 gripper_open()
10
```

3. Stack Cube on Cube: Pick the green cube and stack it on top of the yellow cube

```
1 # Stage 1: Move gripper to green cube
2 gripper_pos = get_centroid("green_cube")
3 move("gripper", "green_cube", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Move green cube above yellow cube
7 yellow_top = get_centroid("yellow_cube")
8 move("green_cube", "yellow_cube", [0, 0, get_height("yellow_cube")])
9 gripper_open()
10
```

4. Move the Soda Can: Pick the soda can from the left table and place it on the right table

```
1 # Stage 1: Move gripper to soda can
2 gripper_pos = get_centroid("soda_can")
3 move("gripper", "soda_can", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Move soda can to right table
7 right_table_pos = get_centroid("right_table")
8 move("soda_can", "right_table", [0, 0, 0])
9 gripper_open()
10
```

5. Fill the Tray: Pick the AA battery and place it into the empty slot in the plastic tray

```

1 # Stage 1: Move gripper to AA battery
2 gripper_pos = get_centroid("AA_battery")
3 move("gripper", "AA_battery", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Move battery to tray slot
7 slot_pos = get_centroid("tray_slot")
8 move("AA_battery", "tray_slot", [0, 0, 0])
9 gripper_open()
10

```

6. Insert the USB Drive: Pick the USB drive from the table and insert it into the laptop's USB port

```

1 # Stage 1: Move gripper to USB drive
2 gripper_pos = get_centroid("USB_drive")
3 move("gripper", "USB_drive", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Align USB drive axis with port
7 usb_axis = get_axis("USB_drive")
8 port_axis = get_axis("laptop_USB_port")
9 parallel(usb_axis, port_axis)
10
11 # Stage 3: Insert into port
12 move("USB_drive", "laptop_USB_port", [0, 0, 0])
13 gripper_open()
14

```

7. Assemble the Lego: Pick the 2x4 Lego brick and attach it to the red baseplate, connecting it to two other bricks

```

1 # Stage 1: Move gripper to Lego brick
2 gripper_pos = get_centroid("Lego_2x4")
3 move("gripper", "Lego_2x4", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Move to baseplate position
7 baseplate_pos = get_centroid("red_baseplate")
8 move("Lego_2x4", "red_baseplate", [0, 0, 0])
9 gripper_open()
10

```

8. Place the Ring: Pick the wooden ring and place it onto the vertical post

```

1 # Stage 1: Move gripper to wooden ring
2 gripper_pos = get_centroid("wooden_ring")
3 move("gripper", "wooden_ring", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Move ring above post
7 post_top = get_centroid("vertical_post")
8 move("wooden_ring", "vertical_post", [0, 0, get_height("vertical_post")])
9 gripper_open()
10

```

9. Put the Lid on the Jar: Pick the plastic jar lid and place it on top of the jar

```

1 # Stage 1: Move gripper to jar lid
2 gripper_pos = get_centroid("jar_lid")
3 move("gripper", "jar_lid", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Move lid to jar top
7 jar_top = get_centroid("jar")
8 move("jar_lid", "jar", [0, 0, get_height("jar")])
9 gripper_open()
10

```

10. Hang the Key: Pick the key and hang it on the keyhook by its hole.

```

1 # Stage 1: Move gripper to key
2 gripper_pos = get_centroid("key")
3 move("gripper", "key", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Align key hole with hook
7 key_hole_axis = get_axis("key_hole")
8 hook_axis = get_axis("keyhook")
9 parallel(key_hole_axis, hook_axis)
10
11 # Stage 3: Hang key
12 move("key", "keyhook", [0, 0, 0])
13 gripper_open()
14

```

11. Push the Dice: Push the white dice across the table until it crosses the black line

```

1 # Stage 1: Move gripper to dice
2 gripper_pos = get_centroid("white_dice")
3 move("gripper", "white_dice", [0, 0, 0])
4
5 # Stage 2: Push dice toward black line
6 line_pos = get_centroid("black_line")
7 dice_pos = get_centroid("white_dice")
8 offset = [line_pos.x - dice_pos.x, 0, 0]
9 move("white_dice", "white_dice", offset)
10

```

12. Flip the Pancake: Use the spatula to flip the pancake in the frying pan

```

1 # Stage 1: Move spatula under pancake
2 spatula_pos = get_centroid("spatula")
3 pancake_pos = get_centroid("pancake")
4 move("spatula", "pancake", [0, 0, -get_height("pancake")])
5
6 # Stage 2: Rotate spatula to flip
7 rotate("spatula", 180)
8
9 # Stage 3: Place pancake back
10 move("pancake", "frying_pan", [0, 0, 0])
11

```

13. Close the Drawer: Push the kitchen drawer closed using the flat of the gripper

```

1 # Stage 1: Move gripper to drawer front
2 gripper_pos = get_centroid("drawer_front")
3 move("gripper", "drawer_front", [0, 0, 0])
4
5 # Stage 2: Push drawer closed
6 closed_pos = get_centroid("drawer_closed_position")
7 move("drawer", "drawer_closed_position", [0, 0, 0])
8

```

14. Press the Doorbell: Press the round, lit doorbell button on the wall

```

1 # Stage 1: Move gripper to doorbell
2 gripper_pos = get_centroid("doorbell_button")
3 move("gripper", "doorbell_button", [0, 0, 0])
4
5 # Stage 2: Press button
6 move("gripper", "doorbell_button", [0, 0, -0.01])
7

```

15. Align the Block: Push the wooden block until it is flush against the corner of the table

```

1 # Stage 1: Move gripper to block
2 gripper_pos = get_centroid("wooden_block")
3 move("gripper", "wooden_block", [0, 0, 0])
4
5 # Stage 2: Push to corner

```

```

6 corner_pos = get_centroid("table_corner")
7 block_pos = get_centroid("wooden_block")
8 offset = [corner_pos.x - block_pos.x, corner_pos.y - block_pos.y, 0]
9 move("wooden_block", "wooden_block", offset)
10

```

16. Scoop the Rice: Use the metal spoon to scoop rice from the pot into the bowl

```

1 # Stage 1: Move spoon into pot
2 spoon_pos = get_centroid("spoon")
3 pot_pos = get_centroid("pot")
4 move("spoon", "pot", [0, 0, -get_height("pot")])
5
6 # Stage 2: Scoop and move to bowl
7 bowl_pos = get_centroid("bowl")
8 move("spoon", "bowl", [0, 0, get_height("bowl")])
9

```

17. Stir the Soup: Use the spoon to stir the liquid in the pot three times clockwise

```

1 # Stage 1: Move spoon into soup
2 spoon_pos = get_centroid("spoon")
3 pot_pos = get_centroid("pot")
4 move("spoon", "pot", [0, 0, -get_height("pot")])
5
6 # Stage 2: Stir clockwise 3 times
7 for _ in range(3):
8     rotate("spoon", -120)
9

```

18. Hammer the Nail: Use the toy hammer to tap the nail until its head is flush with the board

```

1 # Stage 1: Move hammer to nail
2 hammer_pos = get_centroid("toy_hammer")
3 nail_pos = get_centroid("nail")
4 move("toy_hammer", "nail", [0, 0, 0])
5
6 # Stage 2: Tap nail until flush
7 while get_height("nail") > 0:
8     move("toy_hammer", "nail", [0, 0, -0.01])
9

```

19. Screw in the Lightbulb: Pick the lightbulb and screw it into the empty lamp socket

```

1 # Stage 1: Move gripper to lightbulb
2 gripper_pos = get_centroid("lightbulb")
3 move("gripper", "lightbulb", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Align with socket
7 bulb_axis = get_axis("lightbulb")
8 socket_axis = get_axis("lamp_socket")
9 parallel(bulb_axis, socket_axis)
10
11 # Stage 3: Screw in
12 move("lightbulb", "lamp_socket", [0, 0, 0])
13 gripper_open()
14

```

20. Pour the Water: Pick the pitcher and pour water into the empty glass until it is half-full

```

1 # Stage 1: Move gripper to pitcher
2 gripper_pos = get_centroid("pitcher")
3 move("gripper", "pitcher", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Tilt pitcher over glass
7 glass_pos = get_centroid("empty_glass")
8 move("pitcher", "empty_glass", [0, 0, get_height("empty_glass")])
9 rotate("pitcher", -45)

```

```

10
11 # Stage 3: Pour until half full
12 # (Assume sensor feedback or timed pour)
13 import time
14 time.sleep(2)
15 rotate("pitcher", 45)
16 gripper_open()
17

```

21. Uncoil the Rope: Manipulate the coiled rope until it forms a straight line from start to end

```

1 # Stage 1: Move gripper to rope start
2 gripper_pos = get_centroid("rope_start")
3 move("gripper", "rope_start", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Pull rope straight
7 end_pos = get_centroid("rope_end")
8 move("rope_start", "rope_end", [0, 0, 0])
9 gripper_open()
10

```

22. Fold the Washcloth: Fold the small, square washcloth in half

```

1 # Stage 1: Move gripper to washcloth
2 gripper_pos = get_centroid("washcloth")
3 move("gripper", "washcloth", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Fold in half
7 fold_line = get_centroid("washcloth_center")
8 move("washcloth", "washcloth_center", [0, 0, 0])
9 gripper_open()
10

```

23. Open the Bag: Use two grippers to pull the handles of the plastic bag apart

```

1 # Stage 1: Move grippers to handles
2 left_handle = get_centroid("bag_handle_left")
3 right_handle = get_centroid("bag_handle_right")
4 move("gripper_left", "bag_handle_left", [0, 0, 0])
5 move("gripper_right", "bag_handle_right", [0, 0, 0])
6 gripper_close("gripper_left")
7 gripper_close("gripper_right")
8
9 # Stage 2: Pull handles apart
10 move("gripper_left", "gripper_left", [-0.1, 0, 0])
11 move("gripper_right", "gripper_right", [0.1, 0, 0])
12

```

24. Drape the Towel: Drape the hand towel over the horizontal bar

```

1 # Stage 1: Move gripper to towel
2 gripper_pos = get_centroid("hand_towel")
3 move("gripper", "hand_towel", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Drape over bar
7 bar_pos = get_centroid("horizontal_bar")
8 move("hand_towel", "horizontal_bar", [0, 0, get_height("horizontal_bar")])
9 gripper_open()
10

```

25. Route the Cable: Route the USB cable around the two posts in an S-shape

```

1 # Stage 1: Move cable to first post
2 cable_pos = get_centroid("USB_cable")
3 post1_pos = get_centroid("post1")
4 move("USB_cable", "post1", [0, 0, 0])
5

```

```

6 # Stage 2: Route around post1
7 orbit("USB_cable", "post1", 90)
8
9 # Stage 3: Move to post2
10 post2_pos = get_centroid("post2")
11 move("USB_cable", "post2", [0, 0, 0])
12
13 # Stage 4: Route around post2
14 orbit("USB_cable", "post2", -90)
15

```

26. Grasp the Marble: Pick the glass marble from a flat surface

```

1 # Stage 1: Move gripper to marble
2 gripper_pos = get_centroid("glass_marble")
3 move("gripper", "glass_marble", [0, 0, 0])
4 gripper_close()
5

```

27. Grasp the Coin: Pick the single coin from the table

```

1 # Stage 1: Move gripper to coin
2 gripper_pos = get_centroid("coin")
3 move("gripper", "coin", [0, 0, 0])
4 gripper_close()
5

```

28. Re-grip the Screwdriver: Pick the screwdriver by its handle, then place it down and re-grip it by its shaft

```

1 # Stage 1: Grip handle
2 handle_pos = get_centroid("screwdriver_handle")
3 move("gripper", "screwdriver_handle", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Place down
7 table_pos = get_centroid("table")
8 move("screwdriver", "table", [0, 0, 0])
9 gripper_open()
10
11 # Stage 3: Re-grip shaft
12 shaft_pos = get_centroid("screwdriver_shaft")
13 move("gripper", "screwdriver_shaft", [0, 0, 0])
14 gripper_close()
15

```

29. Pick the Book: Pick the paperback book from the shelf by its spine

```

1 # Stage 1: Move gripper to book spine
2 spine_pos = get_centroid("book_spine")
3 move("gripper", "book_spine", [0, 0, 0])
4 gripper_close()
5

```

30. Hook the Mug: Hook a gripper finger through the handle of the coffee mug and lift it

```

1 # Stage 1: Move gripper to mug handle
2 handle_pos = get_centroid("mug_handle")
3 move("gripper", "mug_handle", [0, 0, 0])
4
5 # Stage 2: Hook and lift
6 gripper_close()
7 move("coffee_mug", "gripper", [0, 0, get_height("coffee_mug")])
8

```

31. Place the T-Block: Pick the T-shaped block and insert it into the matching T-shaped slot on the board

```

1 # Stage 1: Move gripper to T-block
2 gripper_pos = get_centroid("T_block")
3 move("gripper", "T_block", [0, 0, 0])
4 gripper_close()

```

```

5
6 # Stage 2: Align with slot
7 block_axis = get_axis("T_block")
8 slot_axis = get_axis("T_slot")
9 parallel(block_axis, slot_axis)
10
11 # Stage 3: Insert into slot
12 move("T_block", "T_slot", [0, 0, 0])
13 gripper_open()
14

```

32. **Assemble the Stack:** Pick the large square block and place it on the table, then place the medium block on it, and finally the small block on top

```

1 # Stage 1: Place large block
2 gripper_pos = get_centroid("large_block")
3 move("gripper", "large_block", [0, 0, 0])
4 gripper_close()
5 table_pos = get_centroid("table")
6 move("large_block", "table", [0, 0, 0])
7 gripper_open()
8
9 # Stage 2: Place medium block
10 gripper_pos = get_centroid("medium_block")
11 move("gripper", "medium_block", [0, 0, 0])
12 gripper_close()
13 move("medium_block", "large_block", [0, 0, get_height("large_block")])
14 gripper_open()
15
16 # Stage 3: Place small block
17 gripper_pos = get_centroid("small_block")
18 move("gripper", "small_block", [0, 0, 0])
19 gripper_close()
20 move("small_block", "medium_block", [0, 0, get_height("medium_block")])
21 gripper_open()
22

```

33. **Plug in the Lamp:** Pick the power plug from the floor and insert it into the wall outlet

```

1 # Stage 1: Move gripper to plug
2 gripper_pos = get_centroid("power_plug")
3 move("gripper", "power_plug", [0, 0, 0])
4 gripper_close()
5
6 # Stage 2: Align plug with outlet
7 plug_axis = get_axis("power_plug")
8 outlet_axis = get_axis("wall_outlet")
9 parallel(plug_axis, outlet_axis)
10
11 # Stage 3: Insert into outlet
12 move("power_plug", "wall_outlet", [0, 0, 0])
13 gripper_open()
14

```

Evaluation by DeepSeek

1. **Sort the Red Cube - CORRECT**
 - Correctly uses gripper movement and basic pick/place
 - Sufficient for the task
2. **Bin the Blue Cylinder - CORRECT**
 - Proper use of height calculation for dropping into bin
 - Clear two-stage approach
3. **Stack Cube on Cube - CORRECT**
 - Correct height calculation for stacking
 - Proper gripper control
4. **Move the Soda Can - CORRECT**

- Simple pick and place between tables
 - Correct implementation
5. **Fill the Tray** - CORRECT
 - Clear battery to slot placement
 - Appropriate for the task
 6. **Insert the USB Drive** - CORRECT
 - Good use of axis alignment with parallel()
 - Proper insertion sequence
 7. **Assemble the Lego** - INSUFFICIENT
 - Missing precise alignment for Lego stud connection
 - No fine positioning for interlocking bricks
 8. **Place the Ring** - CORRECT
 - Correct height calculation for post placement
 - Appropriate for ring-on-peg task
 9. **Put the Lid on the Jar** - CORRECT
 - Proper lid placement using height
 - Sufficient for basic lid placement
 10. **Hang the Key** - CORRECT
 - Good axis alignment for hooking
 - Appropriate for key hanging
 11. **Push the Dice** - CORRECT
 - Correct pushing with offset calculation
 - Proper non-prehensile manipulation
 12. **Flip the Pancake** - INSUFFICIENT
 - Missing spatula grip and manipulation
 - Rotate() on spatula doesn't achieve flipping motion
 13. **Close the Drawer** - CORRECT
 - Proper pushing to closed position
 - Appropriate for drawer closing
 14. **Press the Doorbell** - CORRECT
 - Correct pressing motion with small offset
 - Sufficient for button pressing
 15. **Align the Block** - CORRECT
 - Good offset calculation for corner alignment
 - Proper pushing implementation
 16. **Scoop the Rice** - INSUFFICIENT
 - Missing actual scooping motion
 - No rice transfer between containers
 17. **Stir the Soup** - INSUFFICIENT
 - Rotate() around fixed point doesn't create stirring motion
 - Need circular trajectory around pot center
 18. **Hammer the Nail** - INSUFFICIENT
 - Infinite loop without termination condition
 - Missing actual hammering impact motions
 19. **Screw in the Lightbulb** - INSUFFICIENT
 - Missing rotational motion for screwing
 - Simple placement won't achieve threaded connection
 20. **Pour the Water** - INSUFFICIENT
 - Time-based pouring unreliable
 - Missing water level detection or controlled pour
 21. **Uncoil the Rope** - INSUFFICIENT
 - Single pull motion won't uncoil rope
 - Missing multiple manipulation points

22. **Fold the Washcloth** - **INSUFFICIENT**
 - Moving to center doesn't achieve folding
 - Missing fold line creation and manipulation
23. **Open the Bag** - **CORRECT**
 - Good dual gripper coordination
 - Proper pulling apart motion
24. **Drape the Towel** - **CORRECT**
 - Correct placement over bar
 - Appropriate for draping task
25. **Route the Cable** - **INSUFFICIENT**
 - orbit() function not in vocabulary
 - Missing S-shape path planning
26. **Grasp the Marble** - **CORRECT**
 - Simple and correct grasping
 - Sufficient for marble pickup
27. **Grasp the Coin** - **CORRECT**
 - Proper coin pickup
 - Correct implementation
28. **Re-grip the Screwdriver** - **CORRECT**
 - Clear re-gripping sequence
 - Proper handle to shaft transition
29. **Pick the Book** - **CORRECT**
 - Correct spine grasping
 - Appropriate for book pickup
30. **Hook the Mug** - **CORRECT**
 - Good hooking and lifting sequence
 - Proper mug handle manipulation
31. **Place the T-Block** - **CORRECT**
 - Good axis alignment for T-slot
 - Proper insertion approach
32. **Assemble the Stack** - **CORRECT**
 - Clear sequential stacking with height calculations
 - Proper multi-stage assembly
33. **Plug in the Lamp** - **CORRECT**
 - Good axis alignment for plug insertion
 - Proper electrical plug placement

I.5. Evaluation Prompt

```
Task\{TASK\_DESCRIPTION\}. For each answer, please judge whether it's correct and sufficient to achieve the tasks. Please ignore lack of force and lack of validation issues. Organize your output in Latex with enumerate.
```