Grounded Vision-Language Interpreter for Integrated Task and Motion Planning

Jeremy Siburian[1,2*] **Keisuke Shirai**[1*] **Cristian C. Beltran-Hernandez**[1*]
Masashi Hamaya[1] **Michael Görner**[3] **Atsushi Hashimoto**[1]
[1]OMRON SINIC X Corporation [2]The University of Tokyo [3]University of Hamburg
https://omron-sinicx.github.io/ViLaIn-TAMP

Abstract: While recent advances in vision-language models (VLMs) have accelerated the development of language-guided robot planners, their black-box nature often lacks safety guarantees and interpretability crucial for real-world deployment. Conversely, classical symbolic planners offer rigorous safety verification but require significant expert knowledge for setup. To bridge the current gap, this paper proposes ViLaIn-TAMP, a hybrid planning framework for enabling verifiable, interpretable, and autonomous robot behaviors. ViLaIn-TAMP comprises three main components: (1) a Vision-Language Interpreter (ViLaIn) adapted from previous work that converts multimodal inputs into structured problem specifications using off-the-shelf VLMs without additional domain-specific training, (2) a modular Task and Motion Planning (TAMP) system that grounds the specifications into actionable trajectory sequences through symbolic and geometric constraint reasoning and leverages learning-based skills for complex manipulation, and (3) a corrective planning module which receives concrete task or motion failure feedback from the TAMP component and feeds adapted logic and geometric feasibility constraints back to ViLaIn to refine the specifications. Evaluated on five challenging manipulation tasks in a cooking domain, ViLaIn-TAMP outperforms direct VLM-as-a-planner approaches by more than 15% mean success rate, increasing interpretability. Finally, ViLaIn-TAMP 's closed-loop architecture exhibits a more than 30% higher mean success rate compared to without corrective planning, improving execution robustness.

Keywords: Vision-Language Models, Task and Motion Planning, Interpretability

1 Introduction

Robots in human-centered environments are expected to plan and execute diverse, complicated tasks autonomously and safely. Instructing such robots through natural language is the long-standing dream of artificial intelligence and robotics research, as natural language not only supports robot execution by non-experts but also enables people to control robots intuitively. These robots are expected to interpret linguistic instructions and make feasible plans by perceiving the environment. In the field of task and motion planning (TAMP), various methods have been proposed to realize autonomous planning and execution by robots [1, 2]. However, they require significant expert knowledge and a time-consuming setup for every task, rendering them impractical for a natural-language instructed robot that must flexibly handle variations of the same task as users might request. Large language models (LLMs) [3, 4, 5] and vision language models (VLMs) [6, 7, 8], which are LLMs empowered with vision capabilities, have the potential to alleviate this barrier. In recent years, research on LLM-based planning has gained attention [9, 10, 11, 12]. The LLM/VLM-as-a-planner approach, which directly converts linguistic instructions (and visual observation) into plans, is a representative example, and methods have been proposed in various domains and task settings [13, 14, 15]. However,

[*]Equal contribution. Contact: cristian.beltran@sinicx.com

Workshop on Safe and Robust Robot Learning for Operation in the Real World at CoRL 2025.

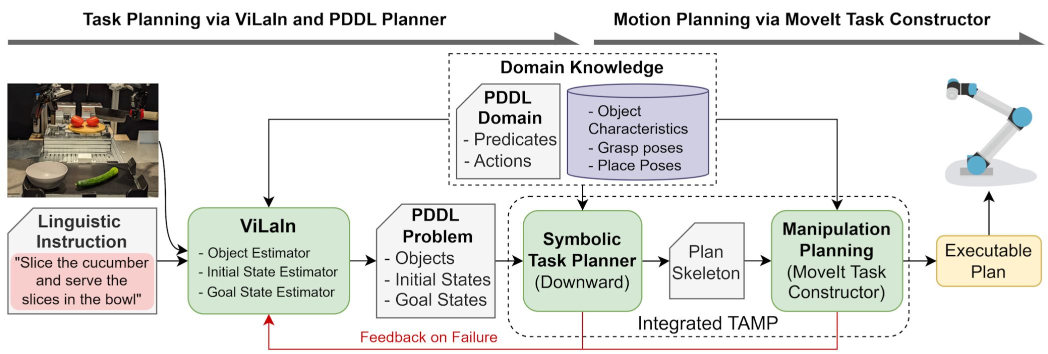

Figure 1: **Overview of ViLaIn-TAMP framework**. Given a linguistic instruction and scene observation, the ViLaIn module generates a PDDL problem, which the integrated TAMP planner solves as a sequence of symbolic actions and collision-free motion trajectories. If successful, the plan is executed on the robot; otherwise, failures are fed back to ViLaIn for re-prompting, revision, and replanning.

their black-box nature has fundamental flaws, notably the missing logical validation and unclear physical feasibility of generated plans. Vision-Language Interpreter (ViLaIn) [16], a hybrid LLM and symbolic planner approach, was proposed to overcome these problems. ViLaIn first uses LLMs to generate human-interpretable problem specifications in PDDL [17] and then triggers a symbolic planner to find systematically verified plans. Nonetheless, ViLaIn is limited to task planning and does not address integrating its outputs with motion planning or executing plans on a real robot.

We propose ViLaIn-TAMP, a novel framework that grounds ViLaIn in TAMP. As shown in Fig. 1, ViLaIn-TAMP first converts linguistic instructions and scene observations into PDDL problems, finds plans by invoking a PDDL task planner [18], verifies its feasibility via a motion planner [19], and finally executes the obtained plans on the real robot. We develop a corrective planning module incorporating error feedback from task and motion planners to address situations where planning fails, such as when PDDL problem descriptions are syntactically incorrect or when logical or geometrical constraints are unsatisfiable. This module enables the system to identify and rectify planning issues, improving overall robustness and reliability. To evaluate ViLaIn-TAMP, we design five real-world manipulation tasks in the cooking domain and demonstrate strong performance of ViLaIn-TAMP both in simulation and on the real robot.

Our contributions are twofold: (i) We propose ViLaIn-TAMP, a novel framework that converts multimodal inputs into PDDL problems and verifies the problems with an integrated TAMP system. (ii) We propose a corrective planning approach that revises erroneous scene-aware PDDL problems based on feedback from the planners. Evaluation results show that ViLaIn-TAMP outperforms a VLM-as-a-planner approach and that corrective planning increases the mean success rate by 30%.

2 Related Work

Foundation Models in Robot Planning. Research and development of large language models (LLMs) [3, 4, 5, 20] and vision language models (VLMs) [6, 7, 8, 21] have been actively studied and developed in natural language processing and computer vision. In robotics, the generalization capabilities and web-scale knowledge of these models have led to the development of innovative methods [9, 22, 11, 10, 23, 12, 24, 25, 26, 27]. Especially in robot task planning, researchers have developed methods that directly generate action sequence plans [28, 10, 12, 13, 16]. Prior work showed LLM's strong capability in symbolic planning [28, 11, 12], developed mechanisms for replanning from errors using LLMs [28, 10, 14], and realized multimodal planning using VLMs [29, 15]. Despite the research efforts, the nature of directly generating plans raises issues such as infeasible plans and the lack of interpretability. To tackle these issues, recent work proposed a hybrid approach of an LLM/VLM and symbolic planner [30, 31, 16]. This approach first generates a problem specification, which is interpretable for humans, and then triggers the planner to find feasible plans. LLM+P [30] showed the LLM's capability of translating linguistic descriptions into PDDL problem specifications, while Xie et al. (2023) [31] generated PDDL goals from linguistic instructions. Shirai et al. (2024) [16] extended this idea by incorporating multimodality and proposed a Vision-Language Interpreter (ViLaIn) that translates linguistic instructions and scene observations into PDDL problem specifications. In this study, we realize a TAMP system based on this hybrid approach by grounding ViLaIn in task and motion planning.

Integrated Task and Motion Planning (TAMP). TAMP approaches consider long-horizon manipulation tasks with multiple intermediate subtasks [2]. TAMP planning [32, 33, 34, 35] typically involves searching for high-level sequences of symbolic actions called plan skeletons and sampling for the low-level motion plans that must be executed to achieve a given goal. Although using large pre-trained models as high-level task planners in TAMP shows promising capabilities [36, 14, 37], they do not have access to lower-level geometric reasoning, such as whether a robot can reach a particular configuration in a collision-free trajectory. To this end, grounding VLMs/LLMs with feedback from motion planning is a promising direction. For instance, [14, 38] implemented motion planning failure reasoning using LLMs/VLMs to correct robot behaviors, focusing on tasks using a single robot arm. In this study, our framework exploits the detailed feedback and failure introspection capabilities of the MoveIt Task Constructor framework for manipulation planning [19]. Although these feedback structures were designed for human users, condensed variants of the feedback are fed back to ViLaIn to refine and correct flawed PDDL problems. Recent works have also combined TAMP with learning methods to perform more complex, contact-rich manipulation skills [39, 40, 41]. Similarly, we augment our TAMP framework with learning-based manipulation skills necessary for real-world cooking tasks, such as slicing using Reinforcement Learning (RL) [42].

3 Preliminaries

PDDL Formulation. We use the Planning Domain Definition Language (PDDL) [17], a standardized and human-readable planning language, to formalize target tasks into a symbolic representation, consisting of a *domain* and *problem*. A PDDL *domain* is formalized by a set of predicates $\mathcal{P}$ and actions $\mathcal{A}$. A PDDL *problem* can be represented as a tuple $(\mathcal{O}, \mathcal{I}, \mathcal{G})$ which is defined by a set of initial objects $\mathcal{O}$, an initial state $\mathcal{I}$, and a goal state $\mathcal{G}$. We use predicates to describe *states*, which can be represented *literals*. For example, we use `(At ?obj ?loc)` to specify that an object `?obj` is currently at location `?loc`. We also model geometric constraints using predicates such as `(CanNotReach ?robot ?obj ?loc)`, to indicate that a robot cannot reach an object at a particular location. In our Cooking domain, we design several types of *actions* that can induce changes to states, such as `pick`, `place`, `equip-tool`, `fixture`, `slice`, `clean-up`, and `serve-food`. Following previous work [43, 44], we provide natural language descriptions of each predicate and action in the domain. These descriptions play a significant role in replacing in-context learning examples. A full description of our PDDL Cooking Domain is provided in Section A.2.

Multimodal Planning Problem Specification. We use ViLaIn to convert multimodal inputs into PDDL problems. This task is formulated as a multimodal planning problem specification, following previous work [16]. The inputs are represented as $(\mathcal{L}, \mathcal{S}, \mathcal{D})$. $\mathcal{L}$ is a linguistic instruction, S is a scene observation, and $\mathcal{D}$ is domain knowledge, which includes a PDDL domain $(\mathcal{P}, \mathcal{A})$ and other domain-specific features (e.g., object characteristics). The output is a PDDL problem $(\mathcal{O}, \mathcal{I}, \mathcal{G})$. The goal of this task is to define $\mathcal{M} : (\mathcal{L}, \mathcal{S}, \mathcal{D}) \rightarrow (\mathcal{O}, \mathcal{I}, \mathcal{G})$, and we employ ViLaIn as $\mathcal{M}$.

4 ViLaIn-TAMP

The ViLaIn-TAMP framework has three major components: 1) a *vision-language interpreter* for PDDL problem generation, 2) a *sequence-before-satisfy* TAMP module for finding symbolically complete, collision-free action plans based on the generated problems, and 3) a *corrective planning* module for refining outputs based on grounded failure feedback. Fig. 1 presents an overview of our framework. The following sections describe each major component of ViLaIn-TAMP in detail.

4.1 Vision-Language Interpreter

Vision-Language Interpreter (ViLaIn) converts a linguistic instruction, scene observation, and domain knowledge $(\mathcal{L}, \mathcal{S}, \mathcal{D})$ into a PDDL problem $(\mathcal{O}, \mathcal{I}, \mathcal{G})$. As shown in Fig. 2, ViLaIn consists of three modules, each generating $\mathcal{O}$, $\mathcal{I}$, and $\mathcal{G}$ of the PDDL problem, respectively. This section describes each module.

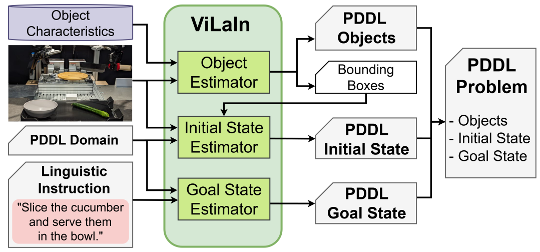

Figure 2: **Overview of ViLaIn.** ViLaIn consists of the three modules that generate a PDDL problem.

The Object Estimator: takes $\mathcal{S}$ and the object characteristics of $\mathcal{D}$ as input and estimates the relevant objects $\mathcal{O}$ of a PDDL problem. The object characteristics are a list of objects of interest with descriptions (e.g., "plate is a white round plate"). The estimator uses a VLM to generate bounding boxes with object labels directly. $\mathcal{O}$ is automatically created based on the detected object labels. The bounding boxes are used with the labels in the subsequent initial state estimation.

The Initial State Estimator: takes a PDDL domain $(\mathcal{P}, \mathcal{A})$, $\mathcal{S}$, and bounding boxes with object labels as input and estimates the initial state $\mathcal{I}$ of a PDDL problem. The estimator feeds the inputs to a VLM and generates $\mathcal{I}$ directly. While the PDDL domain contains $\mathcal{P}$ and $\mathcal{A}$, we only use the former as input.

The Goal State Estimator: takes a PDDL domain $(\mathcal{P}, \mathcal{A})$ and $\mathcal{L}$ and generates the goal state $\mathcal{G}$ of a PDDL problem. For the PDDL domain, we only use $\mathcal{P}$, as with the initial state estimator.

4.2 Integrated Task And Motion Planning

After a complete PDDL problem is generated by the ViLaIn module, an integrated TAMP module is used to search for a sequence of actions and their corresponding motion trajectories. This section describes the task-level and manipulation-level planning frameworks used and their integration with learning-based skills.

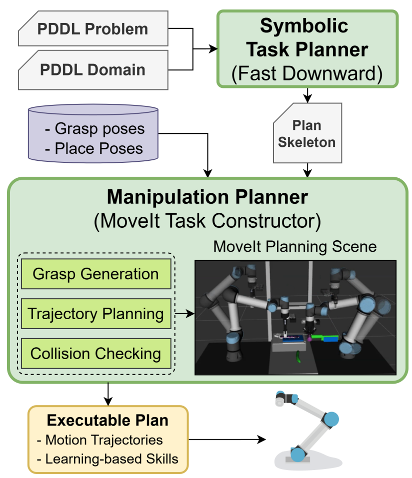

Figure 3: Overview of integrated task and motion planning framework.

Symbolic Task Planning: For task-level planning, we leverage the off-the-shelf symbolic planner Fast Downward [18], which supports PDDL specifications.

Multi-step Manipulation Planning: In complex manipulation tasks, certain stages of a task are often strongly interrelated and cannot be considered independently of each other, particularly for dual-arm manipulation. For example, in the context of our slicing task, certain grasp poses for fixturing an object by an arm may interfere with slicing and cause collisions with the other arm. Conventional motion planners such as MoveIt [45] are typically restricted to a single-arm pick-and-place pipeline, with limited functions for dual-arm coordination [46]. To alleviate this issue, we use the MoveIt Task Constructor (MTC) framework [19], an open-source manipulation planning framework for multi-step tasks. The framework resolves interdependencies between different manipulation phases through controlled co-parameter sampling, supporting finite appropriate attempts for adequate coverage of each symbolic plan.

Integrated TAMP Module: We integrated symbolic task planning and manipulation planning using a *sequence-before-satisfy* strategy [2]. As shown in Fig. 3, we first search for a high-level *sequence* of actions using Fast Downward. After a plan skeleton is found, we use MTC to sample low-level motions that *satisfy* the plan, mapping abstract actions to MTC *stages*. Additionally, we integrated learning-based methods into the TAMP framework to handle complex skills. Classical motion planning alone is insufficient for executing contact-rich manipulation skills, such as food slicing, which demand rapid adaptation to environmental changes. Instead, learning-based methods are better suited for these tasks [42]. We implemented such skills as mock MTC stages to integrate them within TAMP. This approach enables seamless transitions between learned skills and primitive trajectories by enforcing start and end pose constraints before and after a skill execution. During planning, only those trajectories that satisfy these pose constraints are considered valid. During execution, we assume the skill reliably returns the robot to the predefined ending pose.

4.3 Corrective Planning (*Re-prompting, Revision, Replanning*) from Failure Feedback

A key challenge in using VLMs for planning is the verification of their outputs. Erroneous outputs may result in failures, either at the task-level due to contradictory problems or at the motion-level due to collisions or reachability constraints. To ground VLMs with failure reasoning, we implement a 3-step corrective planning (CP) approach, which involves *re-prompting* the model with the failure feedback to *revise* the PDDL problem, and then *replanning* using the revised PDDL problem. We denote the failure feedback as $\mathcal{E}$, which contains the feedback from the task planner or from MTC.

Grounding VLMs with Motion Failure Feedback. A key advantage of the MTC module is that it provides detailed feedback and failure introspection capabilities. When motion planning fails, the MTC returns feedback containing the following three components: 1) **default failure comments** provided by the MTC planning module, 2) the **task execution trace**, which indicates at which step failure occurred, and 3) a **synthesized message** that extracts semantically meaningful motion-level feedback from both the comments and trace. We can extract this detailed failure feedback and evaluate the impact of each component for reasoning on manipulation planning failures. Example outputs of each MTC feedback component and their implementation details can be found in Appendix A.8.

5 Experimental Evaluation

We aim to answer the following questions through our experiments and evaluation: $\mathcal{Q}$**1**: How effective is ViLaIn-TAMP compared to a VLM-as-a-planner approach of directly generating action sequences? $\mathcal{Q}$**2**: How effective is the proposed corrective planning module? $\mathcal{Q}$**3**: Are in-context learning examples necessary for achieving high success rates? and $\mathcal{Q}$**4**: What kind of failure modes often occur that ViLaIn-TAMP may struggle with?

5.1 Experiment Setting

Tasks. To validate our proposed method, we perform experiments on five manipulation tasks in the cooking domain; The full details of each task (scene configuration and linguistic instruction) are provided in Section A.3.

- **Pick and Place**: The goal is to move a target object to a desired location.
- **Pick Obstacles Dual Arm**: The goal is to move a target object to a desired location. However, the target location is occupied by another object. We allow the system to utilize both robot arms in the initial state. In a *dual-arm strategy*, one robot removes the collision object out of the way, while another robot performs pick and place with the target object.
- **Pick Obstacles Single Arm**: The goal is the same as above. However, in this task, we prompt the model to use only one arm. In a *single-arm strategy*, the robot must remove the obstacle object, place it in another collision-free location, and then perform a pick and place for the target object to the target location.
- **Slice Food**: The goal is to slice a food item (e.g., fruit or vegetable) using a tool (e.g., knife). We adopt a dual-arm approach where a robot grasps and fixtures the object onto the cutting board, while another robot performs the slicing operation. A key aspect of the task is that specific choices in which robot to use for which action may result in infeasible plans (e.g., collisions between robots), highlighting the interrelated nature of the task.
- **Slice and Serve**: We extend the Slice Food task by adding another goal of serving the vegetable/fruit slices in a desired location (e.g., a bowl or plate).

We created 10 problems for **Pick and Place**, **Slice Food**, and **Slice and Serve**. For the **Pick Obstacles tasks**, we created 9 problems common to **Single Arm** and **Dual Arm**.

Model Configurations. To evaluate our framework and show the effectiveness of corrective planning (CP), we consider the following model configurations:

- **ViLaIn-TAMP-CP**: Our proposed framework with CP. The maximum number of CP attempts is controlled by $N_{\text{CP, max}}$ (e.g., $N_{\text{CP, max}} = 3$ allows CP for three times).
- **ViLaIn-TAMP-No-CP**: Our proposed framework without CP.
- **Baseline-CP**: This approach uses LLMs/VLMs to directly generate action plans with CP. The model takes a set of observation $\mathcal{O}$, instruction $\mathcal{L}$, and PDDL Domain $\mathcal{D}$ as input and directly generate a set of actions $\mathcal{A}$ as output, which are solved by the TAMP module. We denote this approach hereon as the *baseline approach* in comparison to our framework.
- **Baseline-No-CP**: The baseline approach without CP.

Evaluation Metrics. We employ the success rate (%) as our main evaluation metric. A trial is considered successful if (1) a symbolically complete, geometrically feasible (i.e., collision-free) plan is found, and (2) the plan is executed successfully, achieving the target goal. Additionally, we adopt evaluation metrics from [16] to verify the models' PDDL generation capabilities and report the MTC planner's planning times. These additional evaluations can be found in Section A.5.

Foundation Model Choices. ViLaIn-TAMP consists of foundation models. In our experiments, we adopted Qwen-2.5VL-7B-Instruct [21] for the object estimator[†] and GPT-4o [6] for the other modules and the corrective planning. The results of using open-source models instead of GPT-4o are discussed in Section A.6. It should be noted that the purpose of our framework is not to strictly compare the performance of various models, but rather to showcase the generalizability of our framework. In object detection, we assumed that the positions of fixed objects (e.g., robot arms and the cutting board) appearing throughout the tasks are known and focused on detecting objects (food ingredients and other tools) that change depending on the task.

Generation Strategy. ViLaIn-TAMP uses a chain-of-thought (CoT) prompting [47, 48] when generating PDDL problems and performing corrective planning. Inspired by previous work [12], we implement a CoT summarization step, where the model is prompted to summarize the domain and propose a generation strategy. We implement CoT for both ViLaIn-TAMP and the baseline approach. As we will see in Sect. 5.2, CoT consistently improves performance and enables LLMs/VLMs to estimate outputs without relying on in-context learning (ICL), which includes input-output examples in prompts to describe tasks. Thus, ViLaIn-TAMP does not use ICL by default. When using ICL, input-output examples are randomly selected from the same type of task. In experiments, we generate 5 outputs for each problem by sampling. Thus, for a task of 10 problems, 50 trials are evaluated. For prompt templates used in the experiments, see Section A.1.

Planning Setup. Algorithm 1 shows ViLaIn-TAMP's planning and execution process. We first generate an initial problem description P_{initial} using ViLaIn, which is then passed on to the TAMP module to find a feasible plan π. If planning fails, ViLaIn revises P_{initial} based on the error message $\mathcal{E}$ from the TAMP module. After a revised problem description P_{revised} is received, the TAMP planner is called again to search for π. Throughout each revision, we collect the history of revised problems $\mathcal{H}_{\mathcal{P}}$ and previous error feedbacks $\mathcal{H}_{\mathcal{E}}$ to maintain context within the model. Planning is deemed successful if a complete plan π is found within $N_{\text{CP, max}}$.

Algorithm 1 ViLaIn-TAMP Planning and Execution

Input: $\mathcal{D}, \mathcal{S}, \mathcal{L}, N_{\text{CP,max}}$
$\mathcal{O}, \mathcal{I}, \mathcal{G} \leftarrow$ VILAIN-INITIAL$(\mathcal{D}, \mathcal{S}, \mathcal{L})$
$P_{\text{initial}} \leftarrow \{\mathcal{O}, \mathcal{I}, \mathcal{G}\}$
$\pi, \mathcal{E} \leftarrow$ TAMP(P_{initial})
if $\pi =$ SUCCESS **then return** π
for $N_{\text{CP}} = 1$ **to** $N_{\text{CP,max}}$ **do**
  $\mathcal{H}_{\mathcal{P}} \leftarrow [\,], \mathcal{H}_{\mathcal{E}} \leftarrow [\,]$
  $P_{\text{revised}} \leftarrow$ VILAIN-REVISE$(P_{\text{initial}}, \mathcal{E}, \mathcal{H}_{\mathcal{P}}, \mathcal{H}_{\mathcal{E}})$
  $\mathcal{H}_{\mathcal{P}}$*.append*$(P_{\text{revised}})$
  $\mathcal{H}_{\mathcal{E}}$*.append*$(\mathcal{E})$
  $\pi, \mathcal{E} \leftarrow$ TAMP(P_{revised})
  if $\pi =$ SUCCESS **then return** π
EXECUTE-REAL-ROBOT(π)

[†]Due to poor object detection by GPT-4o in preliminary experiments, we used Qwen-2.5VL.

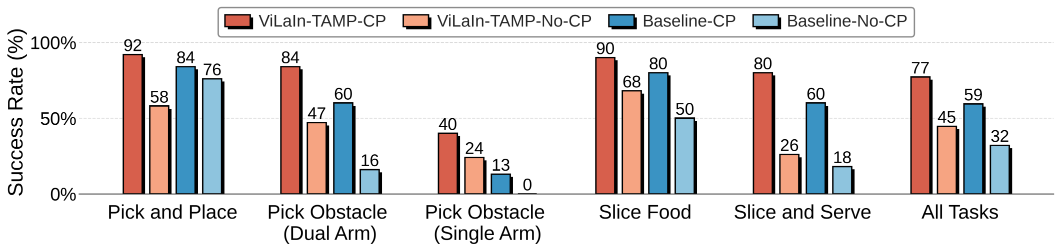

Figure 4: **Comparison of ViLaIn-TAMP and the baseline,** evaluating their performance on five cooking tasks with and without corrective planning (CP). The maximum number of CP attempts is set to 3. ViLaIn-TAMP consistently outperforms the baseline in all tasks. CP is effective in both models, consistently improving the success rate by a large margin.

5.2 Experimental Results

Comparison against Baseline ($\mathcal{Q}$1). As illustrated in Fig. 4, ViLaIn-TAMP-CP achieves the highest success rate across all tasks. Compared to the baseline, ViLaIn-TAMP outperforms the VLM-as-a-planner approach by an average margin of 10%, with even more significant improvements observed as task complexity increases.

Effect of Corrective Planning ($\mathcal{Q}$2). We verify the effect of corrective planning by using different maximum corrective planning attempts ($N_{\text{CP, max}} = 0, 1, 3$). Overall, the results show that ViLaIn-TAMP-No-CP has a lower average success rate of 45%. CP significantly improves all tasks' success rate (Fig. 5).

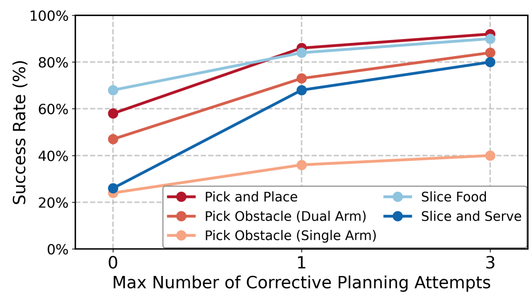

Figure 5: **Effectiveness of Corrective Planning for ViLaIn-TAMP-CP.** Increasing the number of CP attempts consistently improves the success rate.

Effect of In-Context Learning ($\mathcal{Q}$3). We analyze whether providing in-context learning (ICL) examples has a significant impact on ViLaIn-TAMP's performance. Overall, we observe a 5.2% increase in the mean success rate for ViLaIn-TAMP-CP as shown in Tabl. 1. Using ICL examples proves to be more significant in the case of ViLaIn-TAMP-No-CP, as the mean success rate increases from 45% to 55%.

Task Name		Success (%)
Pick and Place		92 (+0)
Pick and Place with Obstacle	Dual Arm	89 (+5)
	Single Arm	49 (+9)
Slice Food		98 (+8)
Slice and Serve		86 (+6)
All Tasks		82 (+5.2)

Table 1: **Effectiveness of In-Context Learning**. Success rate when using ICL in addition to CP.

Failure Modes Analysis ($\mathcal{Q}$4). As shown in Fig. 6, failure modes have been classified as the following: (1) detecting non-existing objects/locations, (2) manipulation planning failure, (3) task planning failures, and (4) successful plans but wrong goal. In mode (1), the model often misrecognizes objects or locations (e.g., an apple is detected as a tomato). A more accurate object detection model may reduce these failures. In mode (2), manipulation planning failures occur due to collision with other objects or between robots. Results show ViLaIn-TAMP reduces this failure type versus the baseline. In some cases, however, the model cannot resolve motion failures, as it misinterprets whether a condition or its negation should be applied. In mode (3), task planning often fails due to incomplete, incorrect, or contradictory use of the predicates. In mode (4), the model may generate a valid task and motion plan for unintended tasks

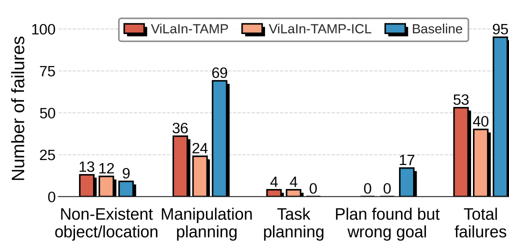

Figure 6: **Failure mode** count per method.

(e.g., the instruction is to move an object to the cutting board, but it moves it to the plate). Failure mode (4) occurs significantly in the baseline method. In contrast, ViLaIn-TAMP, which leverages a symbolic task planner, did not experience such hallucinations, demonstrating the importance of symbolic representation for increasing reliability and predictability.

5.3 Real Robot Validation

We validated ViLaIn-TAMP on a physical robot system comprising two robotic arms (UR5e, Universal Robots A/S, Denmark). Each arm has an incorporated force-torque sensor at its wrist and is equipped with a wrist-mounted RGB-D camera (RealSense D435, Intel Corporation, USA) and a parallel gripper (2F-85 and 2F-140, Robotiq, Canada). The arm allocated for the slicing action has a Robotiq 2F-85 parallel gripper with custom fingertips to equip the kitchen knife, as shown in Fig. 7.

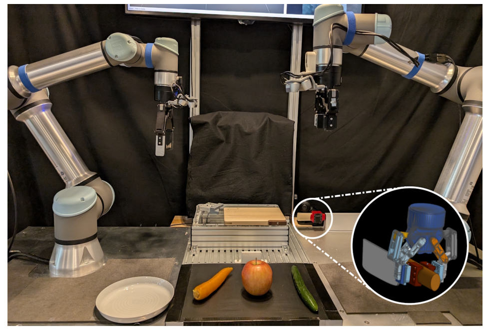

Figure 7: Dual arm robotic system for real-world validation on a cooking domain.

Real-World Perception: During planning, TAMP uses a rough estimation of the object poses, but real-world execution requires a perception module to estimate the object poses accurately. We combine the Qwen2.5-VL-7B-Instruct open-vocabulary detector [21] with the Segment Anything Model 2 [49] for object detection and segmentation. See Sect. A.7 for more details.

Pre-defined Skill Library: For the slicing action, we leverage Reinforcement Learning (RL) and compliance control. A previous study [42] showed that such a method can adapt to unseen objects and requires as little as a single slicing motion to acquire the force profile of the object.

Real-World Implementation: We begin with a valid plan generated by ViLaIn-TAMP. This plan is considered *optimistic* because, although it is based on current visual observations, it may not fully capture positional uncertainties of objects or account for environmental changes caused by the robot's actions (for example, the altered state of an object after slicing). To ensure robust and reliable execution in the real world, we adopt the following approach:

- **Action Classification:** Post-planning, actions are classified as: **(A)** Trackable motion-based actions (e.g., pick/place, equip tool). **(B)** Environment-altering learning skills (e.g., slicing).
- **Plan Segmentation:** The plan is segmented according to this classification, grouping only **(A)** actions for potential replanning.
- **Adaptive Execution:** During execution, actions in **(B)** are carried out sequentially and independently to update our understanding of the environment. Before executing any group of **(A)** actions, we replan them if needed based on the latest state, ensuring the plan remains robust to any changes or uncertainties.

6 Conclusion and Future Work

This work introduced ViLaIn-TAMP, a general-purpose framework integrating VLMs with symbolic and geometric planning to enable robust, interpretable, and autonomous robot manipulation in real-world environments. By leveraging ViLaIn to translate multimodal inputs into structured PDDL problem specifications and coupling this with a TAMP system, our approach systematically verifies generated plans' logical and physical feasibility. Including a corrective planning module, which iteratively refines plans based on feedback from both task and motion planning failures, proved essential for improving reliability, with corrective planning yielding over 30% higher mean success rates across five challenging cooking manipulation tasks compared to direct VLM planning approaches, especially as task complexity increases. In our current framework, we assume that a human-crafted PDDL domain is provided. In future work, we aim to implement automatic PDDL domain generation [44, 50], as well extend to handle unexpected failures during execution time.

References

[1] E. Karpas and D. Magazzeni. Automated Planning for Robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):417–439, May 2020. ISSN 2573-5144, 2573-5144.

[2] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[3] OpenAI, J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, et al. GPT-4 technical report, 2024.

[4] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al. The Llama 3 herd of models, 2024.

[5] M. Abdin, J. Aneja, H. Behl, S. Bubeck, R. Eldan, S. Gunasekar, M. Harrison, R. J. Hewett, M. Javaheripi, P. Kauffmann, et al. Phi-4 technical report, 2024.

[6] OpenAI, A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, et al. GPT-4o system card, 2024.

[7] G. Team, P. Georgiev, V. I. Lei, R. Burnell, L. Bai, A. Gulati, G. Tanzer, D. Vincent, Z. Pan, S. Wang, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024.

[8] H. Liu, C. Li, Q. Wu, and Y. J. Lee. Visual instruction tuning. In *Proceedings of the 2023 Advances in neural information processing systems (NeurIPS)*, volume 36, pages 34892–34916, 2023.

[9] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. In *Proceedings of the 2023 Conference on Robot Learning (CoRL)*, volume 205, pages 1769–1782, 2023.

[10] K. Lin, C. Agia, T. Migimatsu, M. Pavone, and J. Bohg. Text2Motion: From natural language instructions to feasible plans. *Autonomous Robots*, 47(8):1345–1365, 2023.

[11] I. Singh, V. Blukis, A. Mousavian, A. Goyal, D. Xu, J. Tremblay, D. Fox, J. Thomason, and A. Garg. ProgPrompt: Generating situated robot task plans using large language models. In *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530, 2023.

[12] T. Silver, S. Dan, K. Srinivas, J. B. Tenenbaum, L. Kaelbling, and M. Katz. Generalized planning in PDDL domains with pretrained large language models. In *Proceedings of the 2024 AAAI conference on artificial intelligence (AAAI)*, volume 38, pages 20256–20264, 2024.

[13] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan. AutoTAMP: Autoregressive task and motion planning with LLMs as translators and checkers. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6695–6702, 2024.

[14] S. Wang, M. Han, Z. Jiao, Z. Zhang, Y. N. Wu, S.-C. Zhu, and H. Liu. LLM3: Large language model-based task and motion planning with motion failure reasoning. In *Proceedings of the 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12086–12092, 2024.

[15] Z. Yang, C. Garrett, D. Fox, T. Lozano-Pérez, and L. P. Kaelbling. Guiding long-horizon task and motion planning with vision language models. *arXiv preprint arXiv:2410.02193*, 2024.

[16] K. Shirai, C. C. Beltran-Hernandez, M. Hamaya, A. Hashimoto, S. Tanaka, K. Kawaharazuka, K. Tanaka, Y. Ushiku, and S. Mori. Vision-language interpreter for robot task planning. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2051–2058, 2024.

[17] M. Fox and D. Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[18] M. Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[19] M. Görner, R. Haschke, H. Ritter, and J. Zhang. MoveIt! Task Constructor for Task-Level Motion Planning. In *Proceedings of the 2019 IEEE International Conference on Robotics and Automation (ICRA)*, pages 190–196, 2019.

[20] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, et al. DeepSeek-R1: Incentivizing reasoning capability in LLMs via reinforcement learning, 2025.

[21] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025.

[22] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Proceedings of the 2022 Conference on Robot Learning (CoRL)*, volume 164, pages 991–1002, 2022.

[23] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *Proceedings of the 2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500, 2023.

[24] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, et al. Open X-Embodiment: Robotic learning datasets and RT-X models. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903, 2024.

[25] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control, 2024.

[26] K. Kawaharazuka, T. Matsushima, A. Gambardella, J. Guo, C. Paxton, and A. Z. and. Real-world robot applications of foundation models: a review. *Advanced Robotics*, 38(18):1232–1254, 2024.

[27] G. R. Team, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, T. Armstrong, A. Balakrishna, R. Baruch, M. Bauza, M. Blokzijl, et al. Gemini robotics: Bringing ai into the physical world, 2025.

[28] S. S. Raman, V. Cohen, E. Rosen, I. Idrees, D. Paulius, and S. Tellex. Planning with large language models via corrective re-prompting. In *Proceedings of the NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[29] A. Mei, G.-N. Zhu, H. Zhang, and Z. Gan. ReplanVLM: Replanning robotic tasks with visual language models. *IEEE Robotics and Automation Letters*, 9(11):10201–10208, 2024.

[30] B. Liu, Y. Jiang, X. Zhang, Q. Liu, S. Zhang, J. Biswas, and P. Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

[31] Y. Xie, C. Yu, T. Zhu, J. Bai, Z. Gong, and H. Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023.

[32] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the 2020 International Conference on Automated Planning and Scheduling (ICAPS)*, volume 30, pages 440–448, 2020.

[33] J. Siburian, C. C. Beltran-Hernandez, and M. Hamaya. Practical task and motion planning for robotic food preparation. In *Proceedings of the 2025 IEEE/SICE International Symposium on System Integration (SII)*, pages 1229–1234. IEEE, 2025.

[34] A. Curtis, X. Fang, L. P. Kaelbling, T. Lozano-Pérez, and C. R. Garrett. Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances. In *Proceedings of the 2022 International Conference on Robotics and Automation (ICRA)*, pages 1940–1946. IEEE, 2022.

[35] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel. Combined task and motion planning through an extensible planner-independent interface layer. In *Proceedings of the 2014 IEEE international conference on robotics and automation (ICRA)*, pages 639–646. IEEE, 2014.

[36] Y. Ding, X. Zhang, C. Paxton, and S. Zhang. Task and motion planning with large language models for object rearrangement. In *Proceedings of the 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2086–2092. IEEE, 2023.

[37] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. Reid, and N. Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. In *Proceedings of the 2023 Conference on Robot Learning (CoRL)*, volume 229, pages 23–72, 2023.

[38] S. S. Raman, V. Cohen, I. Idrees, E. Rosen, R. Mooney, S. Tellex, and D. Paulius. Cape: Corrective actions from precondition errors using large language models. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14070–14077. IEEE, 2024.

[39] A. Mandlekar, C. R. Garrett, D. Xu, and D. Fox. Human-in-the-loop task and motion planning for imitation learning. In *Proceedings of the 2023 Conference on Robot Learning (CoRL)*, pages 3030–3060. PMLR, 2023.

[40] Z. Wang, C. R. Garrett, L. P. Kaelbling, and T. Lozano-Pérez. Learning compositional models of robot skills for task and motion planning. *The International Journal of Robotics Research*, 40(6-7):866–894, 2021.

[41] Z. Yang, C. R. Garrett, T. Lozano-Pérez, L. Kaelbling, and D. Fox. Sequence-based plan feasibility prediction for efficient task and motion planning. In *CoRL 2022 Workshop on Learning, Perception, and Abstraction for Long-Horizon Planning*, 2022.

[42] C. C. Beltran-Hernandez, N. Erbetti, and M. Hamaya. SliceIt! - a dual simulator framework for learning robot food slicing. In *Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4296–4302. IEEE, 2024.

[43] P. Smirnov, F. Joublin, A. Ceravola, and M. Gienger. Generating consistent pddl domains with large language models. *arXiv preprint arXiv:2404.07751*, 2024.

[44] J. Oswald, K. Srinivas, H. Kokel, J. Lee, M. Katz, and S. Sohrabi. Large language models as planning domain generators. In *Proceedings of the 2024 International Conference on Automated Planning and Scheduling (ICAPS)*, volume 34, pages 423–431, 2024.

[45] D. Coleman, I. Sucan, S. Chitta, and N. Correll. Reducing the barrier to entry of complex robotic software: a MoveIt! case study. *arXiv preprint arXiv:1404.3785*, 2014.

[46] P. M. Fresnillo, S. Vasudevan, W. M. Mohammed, J. L. M. Lastra, and J. A. P. Garcia. Extending the motion planning framework—MoveIt with advanced manipulation functions for industrial applications. *Robotics and Computer-Integrated Manufacturing*, 83:102559, 2023.

[47] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 2022 Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 24824–24837. Curran Associates, Inc., 2022.

[48] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa. Large language models are zero-shot reasoners. *Proceedings of the 2022 Advances in neural information processing systems (NeurIPS)*, 35:22199–22213, 2022.

[49] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, et al. SAM 2: Segment anything in images and videos. *arXiv preprint arXiv:2408.00714*, 2024.

[50] M. Han, Y. Zhu, S.-C. Zhu, Y. N. Wu, and Y. Zhu. InterPreT: Interactive predicate learning from language feedback for generalizable task planning. In *Proceedings of the 2024 Robotics: Science and Systems (RSS)*, 2024.

[51] B. Hui, J. Yang, Z. Cui, J. Yang, D. Liu, L. Zhang, T. Liu, J. Zhang, B. Yu, K. Lu, et al. Qwen2.5-coder technical report, 2024.

[52] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers, 2023.

[53] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, page 611–626, 2023.

Appendix

A.1 Prompts

We use different prompts for each generation task. This section showcases the prompts in the experiments.

Object Detection. The object detection is performed with the following prompt:

```
Given an image showing a robotic experiment environment, detect the following objects:
{objects_of_interest}
Output the bounding boxes for the objects in the form of [xmin, ymin, xmax, ymax].
```

where `{objects_of_interest}` is a list of objects of interest with their brief explanations.

Initial State Generation. The initial state generation is performed with the following prompt:

```
1.   You are an agent for robot task planning. Given a scene observation as an image, objects with types appeared in the environment, and their locations are represented by bounding boxes, you are expected to write the initial state of the environment as a set of predicates. A predicate consists of a predicate name and its arguments, and all written predicates are assumed to be true. Negative predicates (e.g., (not (predicate ...)) do not appear in the initial state. Available predicates are defined as:
2.  {predicates}
3.
4.  ...
5.
6.  ### Example {k} (start)
7.  The objects: {ex_objects}
8.  The bounding boxes: {ex_bounding_boxes}
9.  The corresponding initial state: {ex_initial_state}
10. ### Example {k} (end)

11.  ...
12.
13. The objects: {objects}
14. The bounding boxes: {bounding_boxes}
15. When generating the problem specification, you must follow the following assumptions:
16. 1) For this task, you are allowed to add and use both robots in the initial state. You can use both a bot and b bot.
17. 2) We assume that a location is occupied and is not free if there is at least one object at the location.
18.
19. Could you write a set of predicates of the initial state for the given objects and locations?
20. Let's think step by step.
21. 1) First, write a short summary of this Cooking domain in words.
22. 2) Now, write the initial state for the given objects and locations.
23. The set of predicates are enclosed by '(:init' and ')' so that '(:init (predicate1 ...) (predicate2 ...) ...)'.
```

where `{predicates}` is a set of predicate definitions in a PDDL domain, `{objects}` is the PDDL objects obtained in the objects estimation, and `{bounding_boxes}` is the bounding boxes for the objects.

Lines 6 to 9 provide input-output examples for ICL, where `{ex_objects}`, `{ex_bounding_boxes}`, and `{ex_initial_state}` represent the PDDL objects, bounding boxes, and initial state of the k-th example. These lines are omitted from the prompt when ICL is not employed.

Goal State Generation. The goal state generation is performed with the following prompt:

```
1.  You are an agent for robot task planning. Given a linguistic instruction specifying the task and objects
with types that appeared in the environment, you are expected to write the desired goal conditions as a set of
predicates. A predicate consists of a predicate name and its arguments, and all written predicates are assumed
to be true. The predicates for the goal conditions should be predicates about target objects after completing
the task. A predicate consists of a predicate name and its arguments, and written predicates are assumed to be
true. Available predicates are defined as:
2.  {predicates}
3.
4.  ...
5.
6.  ### Example {k} (start)
7.  The objects: {ex_objects}
8.  The initial state: {ex_initial_state}
9.  The linguistic instruction: {ex_linguistic_instruction}
10. The corresponding goal conditions: {ex_goal_state}
11. ### Example {k} (end)
12.
13. ...
14.
15. The objects: {objects}
16. The initial state: {initial_state}
17. The linguistic instruction: {linguistic_instruction}
18. When generating the problem specification, you must follow the following assumptions:
19. 1) For this task, you are allowed to add and use both robots in the initial state. You can use both a bot
and b bot.
20. 2) We assume that a location is occupied and is not free if there is at least one object at the location.
21.
22. Could you write a set of predicates of the goal conditions for the given instruction and objects?
23. Let's think step by step.
24. 1) First, write a short summary of this Cooking domain in words.
25. 2) Now, write the goal conditions for the given instruction and objects.
26. The set of predicates are enclosed by '(:goal (and' and ')' so that '(:goal (and (predicate1 ...) (predicate2
...) ...))'.
```

where `{predicates}` and `{objects}` are shared with the prompt for initial state generation. `{initial_state}` is the PDDL initial state obtained by the initial state generation. `{linguistic_instruction}` is a linguistic instruction.

Lines 6 to 11 provide ICL input-output examples. Here, `{ex_linguistic_instruction}` and `{ex_goal_state}` are new variables for the k-th example, representing linguistic instruction and the PDDL goal state, respectively. These lines are omitted if ICL is not used.

Corrective Planning for Generated PDDL Problems. Corrective planning for PDDL generation is performed with the following prompt:

```
1.  You are an agent for robot task planning. Given a linguistic instruction and a scene observation as an image, you are expected to write a problem specification that consists of objects, the initial state of the environment, and the desired goal conditions. The initial state and the goal conditions are expressed by predicates. A predicate consists of a predicate name and its arguments, and all written predicates are assumed to be true. Negative predicates (e.g., (not (predicate ...)) do not appear in the initial state. Available predicates are defined as:
2.  {predicates}
3.
4.  With the generated specification, the task planner first finds a sequence of symbolic actions, and the motion planner then finds a sequence of physical actions. The symbolic actions contain preconditions and effects that must be True before and after it is executed, respectively. The actions are defined as:
5.  {actions}
6.
7.  Now you are given the instruction and scene observation.
8.  The instruction: {linguistic_instruction}
9.
10. You created the following problem specification:
11. {pddl_problem}
12. However, planning failed and returned the following feedback:
13. {feedback}
14.
15. When generating the problem specification, you must follow the following assumptions:
16. 1) For this task, you are allowed to add and use both robots in the initial state. You can use both a bot and b bot.
17. 2) We assume that a location is occupied and is not free if there is at least one object at the location.
18.
19. We assume that planning failure occurs because the problem specification is incomplete. Could you generate the revised specification?
20. Let's think step by step.
21. 1) Identify the cause of failure based on the failure feedback.
22. 2) Generate a revised specification based on the cause of failure.
```

where `{actions}` is a list of actions and their definitions in a PDDL domain. `{pddl_problem}` is a previously generated PDDL problem, and `{feedback}` is an error feedback from a planner. Lines 10 to 13 can be repeated if error correction is further needed after previous replanning attempts. Note that we apply the same prompt for replanning in both task and motion planning.

Plan Generation (for Baseline). Plan generation is performed with the following prompt:

```
You are an agent that generates a plan of actions that accomplishes the task specified by a linguistic instruction. Objects for the task are given as a combination of object name and type. Locations for the objects are represented by bounding boxes.
Now the instruction is:
{linguistic_instruction}

The objects are:
{objects}

The locations of the objects by bounding boxes are:
{bounding_boxes}

Available actions are defined as:
{actions}

The actions have preconditions and effects that must be satisfied before and after an action. These are represented by predicates that are defined as:
{predicates}

When generating the plan of actions, you must follow the following assumptions:
1) For this task, you are allowed to add and use both robots in the initial state. You can use both a bot and b bot.
2) We assume that a location is occupied and is not free if there is at least one object at the location.

Output the plan of actions in JSON format without further explanation.
Let's think step by step.
1) First, write a short summary of this Cooking domain in words.
2) Now, generate a plan of actions that accomplishes the task specified by the instruction.
The output must be a list of actions, and the action parameters are selected from the objects.
Each action is a string and the parameters must not be enclosed by "" (e.g., ["action1(argument1, argument2, ...)", ...]).
```

where variable names are shared from the previous prompts.

Corrective Planning for Generated Plan (for Baseline). Corrective planning for plan generation is performed with the following prompt:

```
1.  You are an agent that generates a plan of actions that accomplishes the task specified by a linguistic instruction. Objects for the task are given as a combination of object name and type. Locations for the objects are represented by bounding boxes.
2.
3.  Now the instruction is:
4.  {linguistic_instruction}
5.
6.  The objects are:
7.  {objects}
8.
9.  The locations of the objects by bounding boxes are:
10. {bounding_boxes}
11.
12. Available actions are defined as:
13. {actions}
14.
15. The actions have preconditions and effects that must be satisfied before and after an action. These are represented by predicates that are defined as:
16. {predicates}
17.
18. You generated the following actions:
19. {plan}
20. However, planning failed and returned the following feedback:
21. {feedback}
22.
23. When generating the plan of actions, you must follow the following assumptions:
24. 1) For this task, you are allowed to add and use both robots in the initial state. You can use both a bot and b bot.
25. 2) We assume that a location is occupied and is not free if there is at least one object at the location.
26.
27. Based on the feedback, revise and generate a sequence of actions without further explanation.
28. Let's think step by step.
29. 1) Identify the cause of failure based on the failure feedback.
30. 2) Now, generate a revised sequence of actions based on the feedback.
```

where `{plan}` is a previously generated plan. Other variables in the prompt are shared from prior prompts. Lines 18 to 21 can be repeated for additional error correction after prior attempts. The same prompt is used for both task and motion re-planning.

Additional Prompt for Pick Obstacles Single Arm. In **Pick Obstacles for Single Arm**, models must complete the task with only one arm. To enforce this restriction, we added the following additional instruction to all the prompts except for object detection.

```
When writing the plan of actions, you must follow the following assumptions:
1) For this task, you are only allowed to use one robot. You can use either a_bot or b_bot, but not both.
2) We assume that a location is occupied and is not free if there is at least one object at the location.
```

A.2 PDDL Cooking Domain

In this section, we provide the full Cooking domain defined using PDDL alongside the natural language descriptions of each predicate.

```
(define (domain cooking)
    (:requirements :strips :equality)
    (:predicates
        (Robot ?x) ; This is predicate is used to declare that something is a robot.
        (PhysicalObject ?x) ; This predicate is used to declare that something is a physical object, like a vegetable or fruit
        (Tool ?x) ; This predicate is used to declare that something is a tool (e.g., knife).
        (Location ?x) ; This predicate is used to declare that something is a location (e.g., tray, cutting_board).
        (ToolHolder ?x) ; This predicate is used to declare that a location is a tool holder (e.g., knife holder).
        (isWorkspace ?loc) ; This predicate is used to declare that a location is a workspace (e.g., in cooking, the workspace is the cutting board).
        (HandEmpty ?robot) ; This predicate is used to declare that a robot's hand is empty and not grasping anything.
```

```
        (Equipped ?robot ?tool) ; This predicated is used when a robot is equipped with a tool, such as a
            ↪ knife.
        (CanNotReach ?robot ?obj ?loc) ; This predicate is used to declare if the robot is unable to reach an
            ↪ object.
        (Grasping ?robot ?obj) ; This predicate is used to declare that a robot is grasping an object and the
            ↪ hand is not empty.
        (isFixturing ?robot ?obj) ; This predicate is used to declare that a robot is fixturing an object.
        (isFixtured ?obj) ; This predicate is used to declare that an object is held down (fixtured).
        (isSliced ?obj) ; This predicate is used to declare that an object has been sliced.
        (At ?obj ?loc) ; This predicate is used to declare that an object is at a specific location and
            ↪ occupying the location.
        (Served ?obj ?loc) ; This predicate is used to declare that an object has been served at a specific
            ↪ location after slicing the object.
        (isNotFree ?loc) ; This predicate is used to declare that a location is not free and occupied by an
            ↪ object.
    )

    ; PICK: Pick up an object
    (:action pick
        :parameters (?robot ?obj ?loc)
        :precondition (and
            (Robot ?robot)
            (PhysicalObject ?obj)
            (Location ?loc)
            (not (CanNotReach ?robot ?obj ?loc))
            (At ?obj ?loc)
            (HandEmpty ?robot)
        )
        :effect (and
            (Grasping ?robot ?obj)
            (not (HandEmpty ?robot))
            (not (At ?obj ?loc))
            (not (isNotFree ?loc))
        )
    )

    ; PLACE: Place an object
    (:action place
        :parameters (?robot ?obj ?loc)
        :precondition (and
            (Robot ?robot)
            (PhysicalObject ?obj)
            (Location ?loc)
            (Grasping ?robot ?obj)
            (not (HandEmpty ?robot))
            (not (At ?obj ?loc))
            (not (CanNotReach ?robot ?obj ?loc))
            (not (isNotFree ?loc))
        )
        :effect (and
            (At ?obj ?loc)
            (not (Grasping ?robot ?obj))
            (HandEmpty ?robot)
            (isNotFree ?loc)
        )
    )

    ; EQUIP_TOOL: Pick up a tool
    (:action equip_tool
        :parameters (?robot ?tool ?loc ?obj)
        :precondition (and
            (Robot ?robot)
            (Tool ?tool)
            (isFixtured ?obj)
            (HandEmpty ?robot)
```

```
            (ToolHolder ?loc)
            (At ?tool ?loc)
            (not (CanNotReach ?robot ?tool ?loc))
        )
        :effect (and
            (Equipped ?robot ?tool)
            (not (HandEmpty ?robot))
            (not (At ?tool ?loc))
        )
    )

    ; FIXTURE: Hold down (fixture) the object on the workspace using a robot arm before slicing
    (:action fixture
        :parameters (?robot ?obj ?loc)
        :precondition (and
            (Robot ?robot)
            (PhysicalObject ?obj)
            (At ?obj ?loc)
            (HandEmpty ?robot)
            (not (CanNotReach ?robot ?obj ?loc))
            (isWorkspace ?loc)
        )
        :effect (and
            (not (HandEmpty ?robot))
            (isFixturing ?robot ?obj)
            (isFixtured ?obj)
        )
    )

    ; SLICE: Slice an object
    (:action slice
        :parameters (?robot ?tool ?obj ?loc)
        :precondition (and
            (Robot ?robot)
            (Tool ?tool)
            (Location ?loc)
            (PhysicalObject ?obj)
            (Equipped ?robot ?tool)
            (isFixtured ?obj)
            (isWorkspace ?loc)
            (At ?obj ?loc)
        )
        :effect (and
            (isSliced ?obj)
        )
    )

    ; UNEQUIP_TOOL: Place down & return a tool
    (:action unequip_tool
        :parameters (?robot ?tool ?loc)
        :precondition (and
            (Robot ?robot)
            (Tool ?tool)
            (ToolHolder ?loc)
            (Equipped ?robot ?tool)
        )
        :effect (and
            (not (Equipped ?robot ?tool))
            (At ?tool ?loc)
            (HandEmpty ?robot)
        )
    )

    ; CLEAN_UP: Subroutine of returning leftover foods
    (:action clean_up
```

```
        :parameters (?robot ?obj)
        :precondition (and
            (Robot ?robot)
            (PhysicalObject ?obj)
            (isSliced ?obj)
            (isFixturing ?robot ?obj)
        )
        :effect (and
            (not (isFixturing ?robot ?obj))
            (not (isFixtured ?obj))
            (HandEmpty ?robot)
        )
    )

    ;SERVE_FOOD: Repeated pick-and-place actions for serving slices onto a location (e.g., plate)
    (:action serve_food
        :parameters (?robot ?obj ?loc1 ?loc2)
        :precondition (and
            (Robot ?robot)
            (PhysicalObject ?obj)
            (Location ?loc1)
            (isWorkspace ?loc1)
            (Location ?loc2)
            (isSliced ?obj)
            (not (isFixtured ?obj))
            (HandEmpty ?robot)
            (not (CanNotReach ?robot ?obj ?loc1))
            (not (CanNotReach ?robot ?obj ?loc2))
        )
        :effect (and
            (Served ?obj ?loc2)
            (not (At ?obj ?loc1))
            (not (isNotFree ?loc1))
        )
    )

)
```

Listing 1: PDDL Cooking Domain.

A.3 Task Details

Tabl. 2 shows the descriptions of our five tasks. `{object}` can be "cucumber", "carrot", or "apple." `{location}` can be "plate" or "cutting board."

A.4 Model Compression with Quantization

We performed inference on open-source models such as Qwen2.5-VL [21] and Qwen2.5-Coder-32B [51], which are used in the experiments in Section A.6, using a single GPU of RTX A6000 (48GB). To realize efficient model inference, we applied a model compression technique called post-training quantization to the models. Concretely, we adopted GPTQ [52] and performed INT4 quantization, representing model parameters in 4-bit integers instead of 32-bit floating point numbers. For further efficient computation, the quantized models were deployed on vLLM[‡], a library for faster LLM inference by memory-efficient attention computation based on PagedAttention [53].

A.5 Additional Evaluation

In this section, we report additional metrics to provide a more comprehensive evaluation of our framework. First, to verify the PDDL generation capabilities of the proposed framework, we adopt

[‡]https://github.com/vllm-project/vllm

Task	Explanation
Pick and Place	**Instruction:** “Move the `{object}` to the `{location}`.” **Description:** The robot is required to move `{object}` from the tray to `{location}`.
Pick Obstacles (Dual Arm)	**Instruction:** “Move the `{object}` to the `{location}`.” **Description:** The robot is required to move `{object}` from the tray to `{location}`, but another object is already placed on `{location}`. The robot first needs to move the other object to the tray and then move the `{object}` to the `{location}`.
Pick Obstacles (Single Arm)	**Instruction:** “Move the `{object}` to the `{location}`.” **Description:** The task is the same as Pick Obstacles Dual Arm. The only difference is that the robot must complete the task using only one of the arms.
Slice Food	**Instruction:** “Slice the `{object}`.” **Description:** The robot is required to move `{object}` from the tray to the cutting board and slice them. The slicing target can be two objects.
Slice and Serve	**Instruction:** “Slice the `{object}` and serve them in the bowl.” **Description:** The task is almost the same as Slice Food. The only difference is that the robot is required to serve the sliced food in the bowl.

Table 2: **Explanation of tasks.** We provide the details of each task, such as the instruction provided and the scene configuration used.

Task Name	Without ICL		With ICL = 1	
	$\mathcal{R}_{\textbf{syntax}}(\uparrow)$	$\mathcal{R}_{\textbf{plan}}(\uparrow)$	$\mathcal{R}_{\textbf{syntax}}(\uparrow)$	$\mathcal{R}_{\textbf{plan}}(\uparrow)$
Pick and Place	1.00	0.96	1.00	1.00
Pick Obstacle Single	1.00	0.82	1.00	0.87
Pick Obstacle Dual	1.00	0.91	1.00	1.00
Slice Food	1.00	0.96	1.00	0.98
Slice and Serve	1.00	0.96	1.00	0.93

Table 3: **Additional Evaluation Metrics for Experiments using GPT-4o.** For each task, we report two evaluation metrics related to the model’s PDDL generation capabilities adopted from [16].

Task Name	Without ICL				With ICL = 1			
	Avg. # Task Planning	Avg. # Motion Planning	Planning Times (sec)		Avg. # Task Planning	Avg. # Motion Planning	Planning Times (sec)	
			Mean	Std. Dev.			Mean	Std. Dev.
Pick and Place	1.64	1.24	6.24	0.11	1.50	1.30	6.27	0.14
Pick Obstacle Single	2.76	1.60	13.89	3.32	2.60	1.62	11.75	4.95
Pick Obstacle Dual	2.00	1.64	11.11	2.72	1.56	1.44	9.59	1.66
Slice Food	1.60	1.28	16.41	1.70	1.46	1.30	16.40	1.61
Slice and Serve	2.30	1.80	32.37	10.97	2.02	1.67	30.30	12.31

Table 4: **Additional Planning Metrics for the TAMP module.** For each task, we report the average number of task and motion planning attempts, as well as the mean and standard deviation of planning time, under both conditions: with and without ICL.

similar evaluation metrics from [16]. In particular, we adopt two metrics for evaluating logical correctness: $\mathcal{R}_{\textbf{syntax}}$, which evaluates if a PDDL problem is syntactically correct, and $\mathcal{R}_{\textbf{plan}}$, which evaluates if a PDDL problem can produce valid plans. The results of the PDDL generation evaluation are shown in Tabl. 3. The results show that using GPT-4o always results in syntactically correct PDDL problems, although they do not immediately translate to having valid plans. When given an in-context learning (ICL) input-output example, the results show that this helps in higher $\mathcal{R}_{\textbf{plan}}$, reducing illogical PDDL problems.

We report planning-related metrics for each of the tasks in Tabl. 4. In particular, we report the average task planning attempts, average motion planning attempts, and the planning time statistics for each of the tasks. The results show that adding the ICL input-output examples reduces the average task planning attempts across all tasks, which suggests that the examples help in reducing incorrect or contradictory propositions, increasing consistency, and the success rate of initial generations. This result is aligned with the results in Tabl. 3 that show using the ICL example increases $\mathcal{R}_{\textbf{plan}}$. We also show the mean planning times of the TAMP planner, which shows stable planning durations with relatively low variance. We attribute the high deviations in certain tasks, such as **Slice and Serve**, to the increase of task complexity and longer horizons (e.g., slicing up to two objects).

A.6 Experiments using Open-Source Models

Suffix	Is CP applied?	Is ICL used?
-No-CP		
-CP	✓	
-No-CP-ICL		✓
-CP-ICL	✓	✓

Table 5: **Suffixes for Model Configurations.** We consider four suffixes that specify whether corrective planning (CP) is applied and whether in-context learning (ICL) is used. For example, Phi4-14B-**No-CP-ICL** represents a Phi-4 variant that does not apply CP but uses ICL for generation.

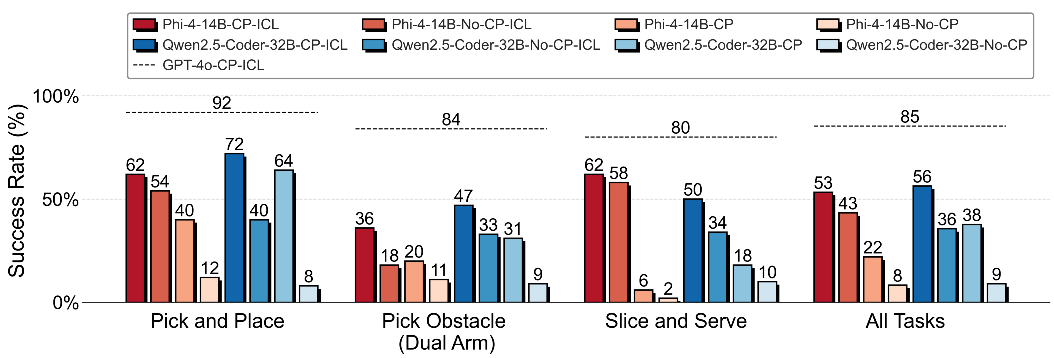

Figure 8: **Evaluation of Open-Source Models.** Model suffixes are explained in Tabl. 5. The results of GPT-4o-CP-ICL are plotted for reference.

Motivated by the dramatic development of open-source models, we evaluate performance when those models are used instead of GPT-4o. We select two models: Microsoft's Phi-4 [5] (14B parameters) and Alibaba's Qwen2.5-Coder-32B-Instruct [51] (32B parameters). Note that these models are text-only ones and that Qwen2.5-VL-7B-Instruct is used as the object estimator as in Section 5. Tabl. 5 shows suffixes to consider four variants based on whether corrective planning (CP) is applied and whether in-context learning (ICL) is used. These suffixes are applied to each model. The open-source models are quantized for efficient inference as described in Section A.4. We select 3 tasks for evaluation: **Pick and Place**, **Pick Obstacles Dual Arm**, and **Slice and Serve**. Similarly to Section 5, we generate 5 outputs for each problem and evaluate the resulting 50 trials per task (45 for Pick Obstacles Dual Arm).

Fig. 8 shows results. The results of GPT-4o-CP-ICL, which is our strongest model from Section 5.2, are shown for reference. We can see that both CP and ICL consistently improve success rates on all tasks. This indicates that using CP and ICL is effective in open-source models as well. The results on all tasks show that the models with CP and ICL achieve more than 50% success rates on our tasks. On the other hand, there is still an about 30% performance gap compared to GPT-4o, suggesting that it remains difficult to achieve competitive performance with open-source models instead of GPT-4o.

A.7 Real-World implementation details

Real-World Perception: During planning, the TAMP module has direct access to the objects' states and poses. However, for execution in the real world, a perception module is necessary to estimate the actual poses of objects. To achieve this, we combine an open-vocabulary object detector (OVOD) and an object segmentation model (OSM)—specifically, Qwen2.5-VL-7B-Instruct [21] and Segment Anything Model 2 [49]. The OVOD uses a linguistic prompt to identify objects of interest within an RGB image, returning their bounding boxes. These bounding boxes are subsequently passed to the OSM, which segments the objects and estimates their orientation relative to the image plane. Finally, using depth information (prior camera calibration is required), we compute the full 6D pose of each object in the robot's coordinate system.

A.8 MoveIt Task Constructor Feedback details

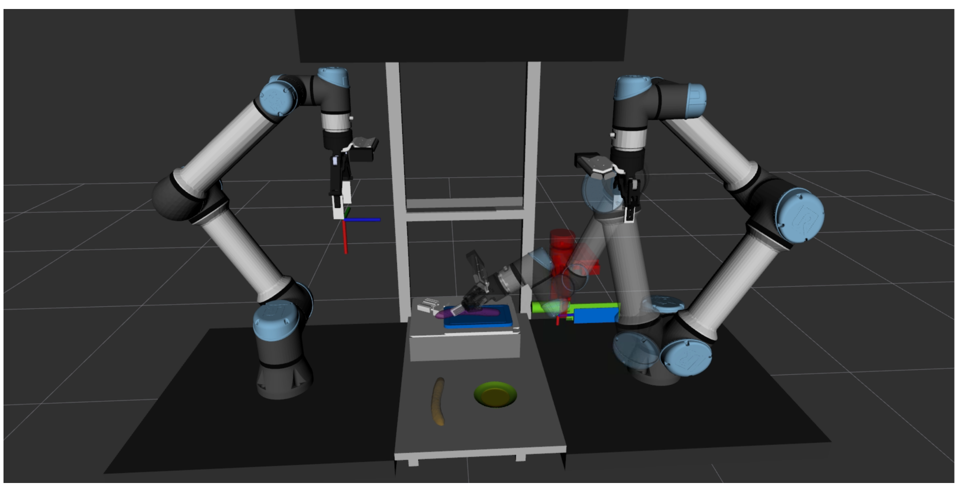

Figure 9: **Example of Motion Planning Failure Visualization in RViz.** In MTC, Both successful and failed motion plans are published and can be visualized in RViz before actual execution, which can be leveraged to extract detailed failure feedback.

In this section, we provide details about the motion planning failure feedback extracted from MoveIt Task Constructor (MTC) [19], as well as examples of each feedback. The examples used are adopted from a failure case in the **Slice Food** task, where there is a collision between a_bot and b_bot when b_bot is in a fixture pose and a_bot attempts to equip the knife. An interactive visualization of the failure in RViz is shown in Fig. 9. One interesting application of this visualization is to have VLMs infer failure causes directly from planning scenes or replayed failed trajectories. In this work, we only extract *natural language* failure feedback and leave visual failure reasoning for future work.

The first type of feedback that can be obtained from MTC is the **default failure comments** that can be provided directly from MTC without significant customization, which reports the *failed MTC stage* and the *failure comments*.

MTC Default Failure Comments

```
MTC Comments:

--- MTC Failure Comments ---
Summary of stages with complete failures:
equip_tool_3 (Pick)
  -> grasp_equip_tool_3 (SimpleGrasp)
    -> compute ik (ComputeIK)
      # of failures: 4
      Failure comments:
        1: eef in collision: a_bot_cam_cables_link - b_bot_forearm_link
        2: Collision between 'a_bot_upper_arm_link' and 'b_bot_wrist_2_link'
        3: eef in collision: a_bot_cam_cables_link - b_bot_forearm_link
        4: Collision between 'a_bot_upper_arm_link' and 'b_bot_wrist_2_link'
```

The second type of feedback is the **task execution trace**, which provides the full task trace and annotates at which step the motion failure occurred.

MTC Task Execution Trace

```
Execution Trace:

--- Task Execution Trace ---
 1) pick b_bot cucumber tray
 2) place b_bot cucumber cutting_board
 3) fixture b_bot cucumber cutting_board
 4) equip_tool a_bot knife knife_holder cucumber [FAILURE]
 5) slice a_bot knife cucumber cutting_board
```

Finally, by leveraging the default failure comments together with the execution trace, we provide a **synthesized failure message** in natural language that describes the failure cause in detail.

MTC Synthesized Failure Message

```
Failure Message:

--- Personalized Failure Message ---
Motion planning has failed.
Motion failed during the 'equip_tool' action.
During motion planning, a_bot failed to equip the knife from the knife_holder.
From the motion planner feedback, the reason for the action failure is because there is a collision
    ↪ between a_bot_cam_cables_link and b_bot_forearm_link
```