Grounded Vision-Language Interpreter for Long-Horizon Bimanual Task and Motion Planning

Jeremy Siburian[1,2,*] Keisuke Shirai[3,*,†] Cristian C. Beltran-Hernandez[1,*,‡]
Masashi Hamaya[1] Michael Görner[4] Atsushi Hashimoto[1]
[1]OMRON SINIC X Corporation, [2]The University of Tokyo, [3]AIST, [4]Constructor University

***Abstract*— While recent advances in vision-language models have accelerated language-guided robot planning, their black-box nature lacks the safety guarantees and interpretability crucial for real-world deployment. Conversely, classical symbolic planners offer rigorous safety verification but require significant expert knowledge for setup. Moreover, most existing methods are limited to single-arm pick-and-place tasks, leaving bimanual manipulation largely underexplored, despite its tightly interdependent subtasks and the need to explicitly manage inter-arm collisions. To bridge this gap, this paper proposes ViLaIn-TAMP, a hybrid planning framework for enabling verifiable, interpretable, and autonomous bimanual robot behaviors. ViLaIn-TAMP comprises three main components: (1) a Vision-Language Interpreter (ViLaIn) adapted from a prior work that converts multimodal inputs into structured PDDL problem specifications, (2) an integrated Task and Motion Planning (TAMP) system that grounds these specifications in actionable trajectory sequences through symbolic and geometric constraint reasoning, explicitly verifying feasibility before execution, and (3) a corrective planning (CP) module which receives structured motion failure feedback and feeds it as constraints back to ViLaIn to refine the specification. We design challenging bimanual manipulation tasks in a cooking domain to evaluate our framework, where experimental results show that ViLaIn-TAMP outperforms a VLM-as-a-planner baseline by 17.5% in mean success rate, with the CP module boosting it further by 32.9%. We further validate ViLaIn-TAMP on a physical dual-arm robotic system. Project page: https://omron-sinicx.github.io/ViLaIn-TAMP**

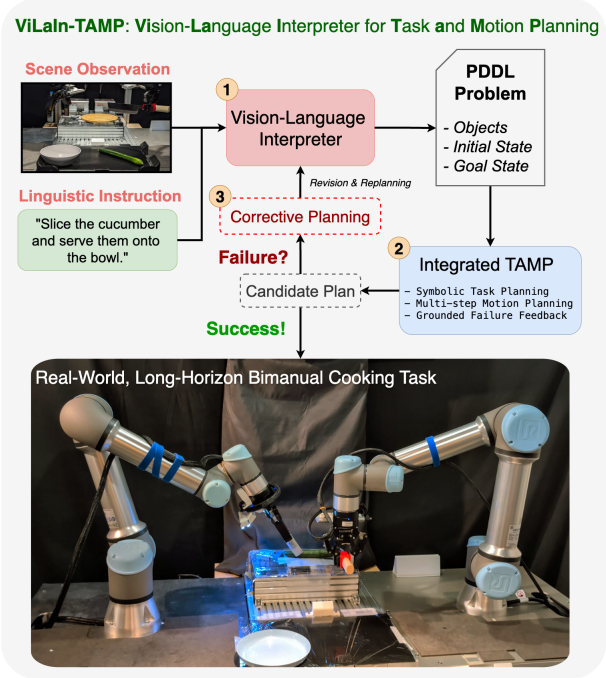


Fig. 1. **ViLaIn-TAMP** is a novel end-to-end planning framework for long-horizon bimanual manipulation that 1) converts multimodal inputs into PDDL problems, 2) finds feasible motion plans via an integrated TAMP system, and 3) reasons over failure feedback for corrective replanning.

I. 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 required to interpret linguistic instructions and make feasible plans by perceiving the environment. Large language models (LLMs) and vision-language models (VLMs) [1], [2], [3] have the potential to alleviate this barrier. In recent years, research on LLM-based planning has gained attention [4], [5], [6]. 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 [7], [8], [9]. However, their black-box nature has fundamental flaws, notably the missing logical validation and unclear physical feasibility of generated plans.

Vision-Language Interpreter (ViLaIn) [10] addresses the former limitation by performing hybrid planning through a symbolic planner. ViLaIn first generates PDDL [11] problem specifications from the instructions and then triggers a symbolic planner to find plans, while syntactically and semantically validating the specifications. Nonetheless, ViLaIn is limited to task planning, and it cannot be directly connected to real robot execution. We bridge this gap while addressing the latter limitation by building a novel framework based on ViLaIn and an integrated TAMP system.

In particular, most existing VLM-based planning methods target single-arm settings, leaving bimanual manipulation

This work was supported by JSPS KAKENHI Grant Number 25K21274.
*Equal contribution. †Work done at OMRON SINIC X Corporation.
‡Corresponding author: cristian.beltran@sinicx.com

and its unique challenges largely underexplored [12], [9]. Consider the dual-arm cooking task shown in Fig. 1 of slicing a vegetable with a knife. This long-horizon task contains subtasks that are strongly interrelated and cannot be considered independently from each other. For instance, one arm must fixture the object while the other slices, and certain grasp poses may cause inter-arm collisions. Furthermore, collisions among grasped objects/tools, workspaces, and robots must be considered for safety, yet most motion planners such as MoveIt [13] are limited to pick-and-place pipelines, have black-box implementations, and ignore interdependence between subtasks [14]. We therefore integrate the MoveIt Task Constructor (MTC) [15] for multi-stage manipulation, and significantly modify it to provide multi-level structured motion failure feedback for VLM reasoning, enabling effective corrective planning for dual-arm tasks.

We propose ViLaIn-TAMP, a novel end-to-end planning framework that *grounds* ViLaIn in TAMP for bimanual manipulation. As shown in Fig. 1, ViLaIn-TAMP first converts linguistic instructions and scene observations into PDDL problems, finds plans by driving a symbolic planner [16], explicitly verifies geometric feasibility via the MTC [15], and finally executes the obtained plans on the real robot. To improve robustness, ViLaIn-TAMP contains a *corrective planning* (CP) module that incorporates failure feedback from the TAMP module to address situations where planning fails, such as when PDDL problems are syntactically incorrect or when logical or geometrical constraints are unsatisfiable. We evaluate ViLaIn-TAMP on five bimanual cooking tasks, demonstrating strong planning capability through quantitative simulation experiments and real-world validation on a physical dual-arm system.

We construct an open-source evaluation dataset covering a custom PDDL domain, problems, linguistic instructions, and scene observations. Evaluated on this dataset, ViLaIn-TAMP shows strong performance over a VLM-as-a-planner baseline with greater interpretability, and CP improves execution robustness. Upon acceptance, we plan to release our dataset, codebase, and custom MTC implementation to support future research and contribute to the robotics and open-source MoveIt community.

II. Related Work

Foundation Models in Robot Planning: Research and development of large language models (LLMs) [17], [2] and vision-language models (VLMs) [1], [3] have been extensively 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 [18]. Especially in robot task planning, researchers have developed methods that directly generate action sequence plans [19], [7]. Prior work showed LLM's strong capability in symbolic planning [5], [6], developed mechanisms for replanning from errors using LLMs [19], [20], [8], and realized multimodal planning using VLMs [7], [9]. Despite extensive research efforts, the nature of directly generating plans raises issues such as infeasible plans and the lack of interpretability. To address these issues, recent work proposed a hybrid approach of combining LLM/VLMs and symbolic planners [21], [22], [10], where a human interpretable specification is first generated before planning. LLM+P [21] showed the LLM's capability of translating linguistic descriptions into PDDL problem specifications, while Xie et al. (2023) [22] generated PDDL goals from linguistic instructions. Vision-Language Interpreter (ViLaIn) [10] extended these ideas by incorporating *multimodality*, enabling whole PDDL problem generation from linguistic instructions and scene observations. However, ViLaIn is currently limited to task planning only and does not consider any real-world robot execution. Like LLM+P, ViLaIn follows a *formalizer* paradigm [21], [23], translating inputs into a PDDL problem under a given domain. We ground this formalizer in an integrated Task and Motion Planning (TAMP) system and close a feedback loop converting motion-level infeasibility into symbolic constraints for re-formalization.

LLMs/VLMs for Task and Motion Planning: TAMP planning [24], [25], [26] typically involves searching for high-level sequences of symbolic actions and sampling for the low-level motion plans that must be executed to achieve a given goal. Recent research has shown promising capabilities in combining large pre-trained models with TAMP [27], [8], [28]. To improve performance, several works utilized LLMs/VLMs to enforce constraints in TAMP planning and replanning from failure feedback [29], [30]. However, [29] is limited to 2D navigation tasks, while [30] does not consider geometric constraints from motion planning feedback. Classical constraint-based TAMP solvers such as PDDLStream [24] and IDTMP [31] propagate geometric infeasibilities as constraints without a foundation model, but assume a correct, fully specified symbolic problem. Here that problem is VLM-produced and may contain misdetected objects, missing predicates, or hallucinated goals; corrective planning thus addresses errors in the problem's *formalization*, not only its motion-level realization. End-to-end approaches such as RT-X [32] and VIMA [12] demonstrate strong performance through large-scale data collection, but lack explicit symbolic planning, making failure introspection and pre-execution verification fundamentally difficult. [8], [33], [34] implemented motion failure reasoning using LLMs/VLMs to correct robot behaviors, but these are limited to single-arm tasks. ViLaIn-TAMP similarly leverages motion failure feedback, but we instead exploit the failure introspection capabilities of the MoveIt Task Constructor framework [15] for dual-arm manipulation, explicitly verifying geometric feasibility before execution, a capability absent in all aforementioned works. While the feedback loop is not arm-specific, its *actionable content* is: reports identify the responsible arm and inter-arm collision pairs, letting the symbolic layer reason about arm-role assignment (which arm fixtures vs. slices), absent in single-arm planning. VLM-TAMP [9] is closest to our work, using VLM-generated subgoals refined by a TAMP planner. However, VLM-TAMP is evaluated only in simulation with no real-robot implementation, and provides only collided

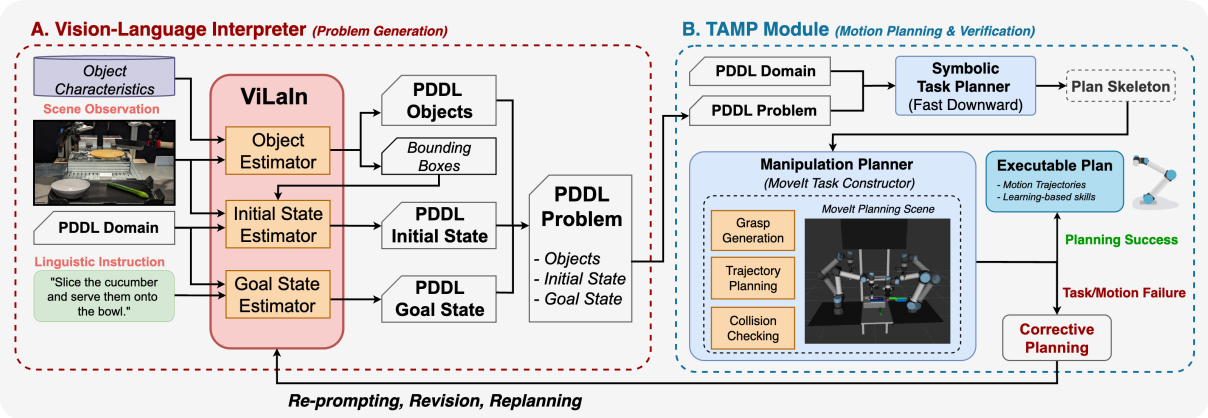


Fig. 2. **Overview of ViLaIn-TAMP framework.** (A) Given a linguistic instruction and scene observation, ViLaIn generates a complete PDDL problem. (B) The TAMP module solves for symbolic actions and collision-free trajectories. If successful, the plan is executed; otherwise, failures are fed back to ViLaIn for revision and replanning.

object names as failure feedback, whereas we extract multi-level structured feedback from our custom MTC implementation for more effective VLM reasoning. Furthermore, while VLM-TAMP and RT-X [32] support dual-arm configurations, the arms operate relatively independently without explicit inter-arm collision reasoning. In contrast, ViLaIn-TAMP handles tightly coupled bimanual coordination (e.g., one arm fixturing while the other slices) with learning-based skills and is validated on a physical dual-arm system.

III. Preliminaries

Task and Motion Planning (TAMP): We adopt the TAMP setting [35], which jointly reasons over a discrete *task* level (which symbolic actions to take, and in what order) and a continuous *motion* level (whether collision-free trajectories realize them). Following common assumptions [35], [36], we assume known object and robot models with approximate object poses from perception, per-action motion samplers (here, MTC), and a collision checker for pre-execution verification. Unlike in pure task planning, a symbolically valid plan must also admit a feasible motion-level realization.

PDDL Formulation: We use the Planning Domain Definition Language (PDDL) [11], 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 as *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`.

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 [10]. 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 $\mathcal{M}$.

IV. Methodology

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. 2 presents an overview of our framework.

A. Vision-Language Interpreter for Problem Generation

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 part A of Fig. 2, ViLaIn consists of three modules, each generating $\mathcal{O}$, $\mathcal{I}$, and $\mathcal{G}$ of the PDDL problem, respectively. While the original ViLaIn and other prior works used ICL input-output examples [10], [21], [30], we instead provide short natural language descriptions of each predicate and action in the domain [37], [38]. These descriptions play a significant role in replacing ICL examples, and we investigate the effects of using ICL examples in our evaluations.

Object Estimator takes $\mathcal{S}$ and the object characteristics of $\mathcal{D}$ as input and estimates the relevant objects $\mathcal{O}$. 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.

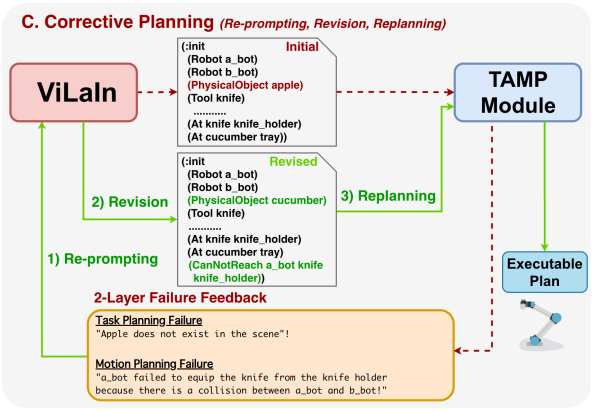


Fig. 3. **Overview of Corrective Planning Module.** ViLaIn-TAMP implements a 3-step corrective planning (CP) approach, which involves 1) *re-prompting* the model with the failure feedback to 2) *revise* the PDDL problem, and then 3) *replanning* using the revised PDDL problem.

Initial State Estimator takes $\mathcal{P}$, $\mathcal{S}$, and bounding boxes with object labels as input and estimates $\mathcal{I}$. The estimator feeds the inputs to a VLM and generates $\mathcal{I}$ directly.

Goal State Estimator takes $\mathcal{P}$ and $\mathcal{L}$ and generates $\mathcal{G}$.

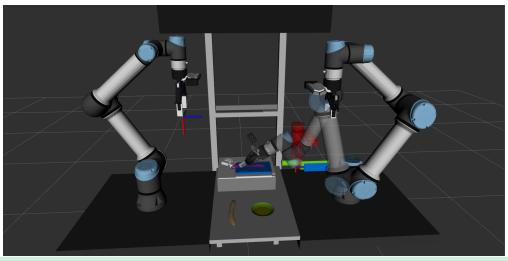
B. Integrated TAMP System

After a complete PDDL problem is generated by ViLaIn, an integrated TAMP module is used to search for a symbolically complete, collision-free action plan.

Symbolic Task Planning: For task-level planning, we leverage the off-the-shelf symbolic planner Fast Downward [16], 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 [13] are typically restricted to a single-arm pick-and-place pipeline, with limited functions for dual-arm coordination [14]. To alleviate this issue, we integrate the MoveIt Task Constructor (MTC) [15], an open-source manipulation planning framework for multi-step tasks, into ViLaIn-TAMP. MTC organizes tasks as a hierarchy of stages within serial containers, where each stage passes intermediate states to adjacent stages, naturally capturing interdependencies between manipulation phases. The framework resolves these interdependencies 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 [35]. As shown in part B of Fig. 2, 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. Each symbolic action is automatically mapped to a corresponding MTC serial container composed of the appropriate stages (e.g., grasp generation, IK computation, trajectory planning, and collision checking), eliminating the need for per-task manual stage specification. Additionally, learning-based skills are integrated into the TAMP framework to handle complex, contact-rich subtasks such as food slicing, which demand rapid adaptation to environmental changes [39]. Following skill-augmented TAMP formulations [40], [41], each learned skill is a black-box operator: it exposes PDDL-style pre/post conditions so the task planner sequences it like a primitive action, while geometrically it is defined only by object-centric start/end poses, keeping its policy-driven trajectory opaque. Skills are realized as mock MTC stages; during planning, only trajectories satisfying these pose constraints are valid.



```
--- MTC Failure Comments ---
Summary of stages with complete failures:
equip_tool_3 (Pick)
  -> grasp_equip_tool_3 (SimpleGrasp)
    -> compute ik (ComputeIK)
      Failure comments: Collision between 'a_bot_upper_arm_link'
            ↪ and 'b_bot_wrist_2_link'
```

```
--- 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
```

```
 --- 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
```

Fig. 4. **Motion Planning Failure Feedback.** Our custom MTC extracts multi-level structured feedback (failure comments, execution trace, and synthesized message) for VLM reasoning. Failed plans can also be visualized in RViz before execution.

C. Corrective Planning from Failure Feedback

The original ViLaIn uses a symbolic planner to validate and revise syntactic errors in the generated PDDL problems. However, this ignores geometrical constraints not captured by the symbolic planner, resulting in motion planning failures due to inter-arm collisions, unreachable objects, and nonexistent objects or locations. These errors can be resolved by obtaining feedback from the motion planner and prompting ViLaIn to revise the PDDLs by adding geometrical constraints (e.g., `(CanNotReach a_bot, knife, knife_holder)`) and modifying incorrect predicates. To realize this feedback loop,

we implement a 3-step corrective planning (CP) approach as shown in Fig. 3, 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 $\mathcal{E}$. In ViLaIn-TAMP, we consider 2-layer feedback from either the task planner or the MTC module.

Extracting Motion Failure Feedback for VLM Reasoning: MTC provides visual introspection of stage-level failures through its RViz panel, but this information is not programmatically accessible. We modify MTC's C++ API to extract failure information after each planning attempt by traversing the task hierarchy and collecting failure comments from nested stages (e.g., collision pairs from IK computation). We then cross-reference these with the symbolic action sequence to identify the responsible action and robot arm. When motion planning fails, the modified MTC module returns feedback at three levels of abstraction: 1) short failure comments from MTC's internal stage data, 2) the task execution trace indicating at which step failure occurred, and 3) a synthesized natural language message constructed from a template populated with the failed action, robot arm, and specific collision or reachability information. Fig. 4 provides an example failure case where a collision occurs between the two robot arms during a food slicing task.

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})
$\mathcal{H}_{\mathcal{P}} \leftarrow [\,]$, $\mathcal{H}_{\mathcal{E}} \leftarrow [\,]$
$N_{\text{CP}} \leftarrow 0$
while $\pi \neq$ SUCCESS **and** $N_{\text{CP}} < N_{\text{CP,max}}$ **do**
  $N_{\text{CP}} \leftarrow N_{\text{CP}} + 1$
  $P_{\text{revised}} \leftarrow$ 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**
  Execute-Real-Robot(π)

V. Experimental Evaluation

Our evaluation addresses five questions: ($\mathcal{Q}1$) how effective ViLaIn-TAMP is versus a VLM-as-a-planner baseline that directly generates action sequences; ($\mathcal{Q}2$) which failure modes it exhibits; ($\mathcal{Q}3$) how much the CP module improves robustness; ($\mathcal{Q}4$) whether ICL examples are necessary; and ($\mathcal{Q}5$) how the foundation-model choice affects performance.

A. Experiment Setting

*1) **Tasks**:* We validate our proposed method on five manipulation tasks in the cooking domain:

- **Pick and Place** aims to move a target object to a desired location.
- **Pick Obstacle Dual Arm** is the same task as Pick and Place except that the location is occupied by another object, causing a collision when directly using the same solution as Pick and Place. One robot removes the collision object (acting as temporary storage) and another arm performs pick-and-place with the target object.
- **Pick Obstacle Single Arm** is the same as Pick Obstacle Dual Arm, but only a single arm is allowed to be used. This task requires more complex planning than Pick Obstacle Dual Arm, as it requires an extra placing action and accurately identifying free placing locations.
- **Slice Food** aims to slice a food item (e.g., fruit or vegetable) using a tool (e.g., knife). A key aspect of the task is that specific choices in which robot to use for certain actions may result in infeasible plans (e.g., collisions between robots), highlighting its interdependency.
- **Slice and Serve** is an extended task of "Slice Food" 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 Obstacle tasks, we created 9 problems common to Single Arm and Dual Arm. All problems are provided with PDDL problems, linguistic instructions, and scene observations. We also created a PDDL domain shared across the tasks.

*2) **Baselines and Comparisons**:* To evaluate our framework and show the effectiveness of corrective planning (CP), we compare the following methods:

- **ViLaIn-TAMP-CP** is 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** is our proposed framework without CP.
- **Baseline-CP** is a VLM-as-a-planner baseline [7], [9] that uses LLMs/VLMs to directly generate action plans with CP, taking $\mathcal{O}$, $\mathcal{L}$, $\mathcal{D}$ as input and directly generating $\mathcal{A}$ as output, which are then solved by the TAMP module (denoted hereon as the *baseline approach*).

We note that end-to-end approaches such as RT-X [32] and VIMA [12] are not directly comparable in our setting, as they do not support PDDL-based symbolic reasoning or pre-execution geometric verification. Since ViLaIn instantiates a similar formalizer paradigm as LLM+P [21], the ViLaIn-TAMP-No-CP variant is a LLM+P-style baseline, isolating the effect of the corrective feedback loop; AutoTAMP [29] instead targets navigation and is not directly applicable.

*3) **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.

*4) **Foundation Model Choices**:* ViLaIn-TAMP consists of foundation models. In our experiments, we adopted Qwen-2.5VL-7B-Instruct [42] for the object estimator* and GPT-4o [1] for the other modules and the corrective planning. 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 only detected objects (foods and other tools) that change depending on the task.

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

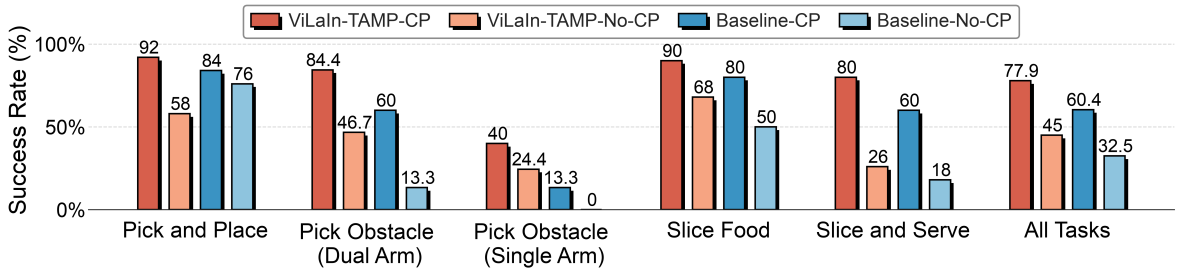


Fig. 5. **Comparison of ViLaIn-TAMP and the baseline,** evaluating their performance on five cooking tasks with and without corrective planning (CP). The maximum number of CP attempts is set to 3. ViLaIn-TAMP consistently outperforms the baseline in all tasks. CP is effective in both models, consistently improving the success rate by a large margin.

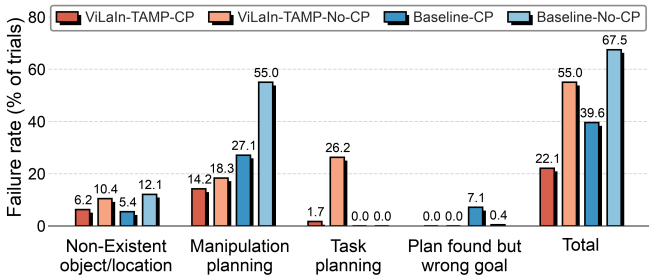


Fig. 6. Failure modes as a rate over all trials.

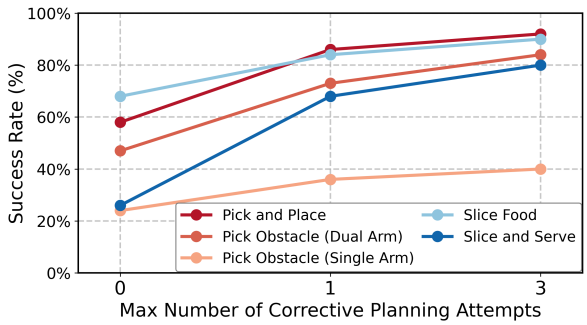


Fig. 7. **Effectiveness of Corrective Planning for ViLaIn-TAMP-CP.** More CP attempts consistently improve the success rate.

*5) **Planning Setup**:* Algorithm 1 shows ViLaIn-TAMP's planning and execution process. We first generate an initial problem P_{initial} using ViLaIn, which is passed to the TAMP module. If planning fails, ViLaIn revises P_{initial} based on failure feedback $\mathcal{E}$, maintaining a history of revisions $\mathcal{H}_{\mathcal{P}}$ and feedbacks $\mathcal{H}_{\mathcal{E}}$ for context. Planning succeeds if a complete plan π is found within $N_{\text{CP, max}}$ attempts.

B. Primary Results

Comparison against Baseline ($\mathcal{Q}$1). As shown in Fig. 5, 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 17.5%, with even more significant improvements observed as task complexity increases. In Tab. I, we report planning-related metrics for each of the tasks, including average task planning attempts, average motion planning attempts, and planning time statistics (mean and standard deviation), denoted as SR, TP, MP, Mean, and SD, respectively. ViLaIn-TAMP is shown to have stable planning durations with relatively low variance. We attribute the high deviations in certain tasks, such as Slice and Serve, to the increase in task complexity and longer horizons (e.g., slicing up to two objects).

Failure Modes Analysis ($\mathcal{Q}$2). 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 (e.g., the instruction is to move an object to the cutting board, but it moves the object 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.

C. Ablation Studies

Effect of Corrective Planning ($\mathcal{Q}$3). We verify the effect of corrective planning by using different maximum corrective planning attempts ($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 the success rate of all tasks (Fig. 7).

Effect of In-Context Learning ($\mathcal{Q}$4). We analyze whether providing ICL input-output examples has a significant impact on ViLaIn-TAMP's performance in Tab. I. Overall, we observe a 5.5% increase in the mean success rate for ViLaIn-TAMP-CP. 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%. The results also show that adding the ICL examples reduces the average task planning attempts across all tasks, which suggests that the examples help in reducing incorrect or contradictory

TABLE I

ADDITIONAL METRICS FOR VILAIN-TAMP WITH AND WITHOUT IN-CONTEXT LEARNING (ICL)

Task Name	Without ICL					With ICL = 1				
	Metrics			Planning Times [sec]		Metrics			Planning Times [sec]	
	%SR	#TP	#MP	Mean	SD	%SR	#TP	#MP	Mean	SD
Pick and Place	92.0	1.64	1.24	6.24	0.11	92.0 (+0.0)	1.50	1.30	6.27	0.14
Pick Obstacle Dual	84.4	2.00	1.64	11.11	2.72	89.0 (+4.6)	1.56	1.44	9.59	1.66
Pick Obstacle Single	40.0	2.76	1.60	13.89	3.32	49.0 (+9.0)	2.60	1.62	11.75	4.95
Slice Food	90.0	1.60	1.28	16.41	1.70	98.0 (+8.0)	1.46	1.30	16.40	1.61
Slice and Serve	80.0	2.30	1.80	32.37	10.97	86.0 (+6.0)	2.02	1.67	30.30	12.31

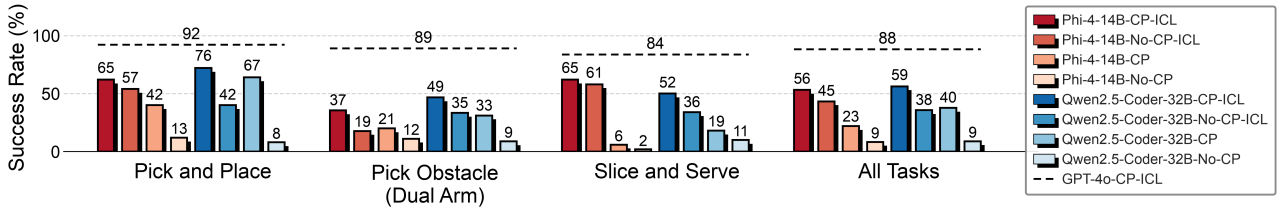


Fig. 8. **Evaluation of Open-Source Models.** The results of GPT-4o-CP-ICL are plotted for reference.

propositions, increasing consistency, and the success rate of initial generations.

Choice of Foundation Models ($\mathcal{Q}$5). In recent years, there has been significant progress in the development of open-source foundation models. Motivated by this, we evaluate performance when those models are used instead of GPT-4o. We select two open-source models: Microsoft's Phi-4 [43] (14B parameters) and Alibaba's Qwen2.5-Coder-32B-Instruct [44] (32B parameters). Note that these models are text-only and that Qwen2.5-VL-7B-Instruct is used as the object estimator. We select 3 tasks for evaluation: Pick and Place, Pick Obstacle Dual Arm, and Slice and Serve. We generate 5 outputs for each problem and evaluate the resulting 50 trials per task (45 for Pick Obstacle Dual Arm). Fig. 8 shows results. The results of GPT-4o-CP-ICL, which is our strongest model from Sect. V-B, are shown for reference. We can see that both CP and ICL consistently improve success rates across all tasks, with CP+ICL exceeding 50% on every task, confirming their effectiveness in open-source models as well. On the other hand, there is still about 30% performance gap compared to GPT-4o, suggesting that it remains difficult to achieve competitive performance with open-source models.

D. Real Robot Validation

Real Robot Setup: Our physical robot system comprises two robotic arms (UR5e). Each arm has an incorporated force-torque sensor at its wrist and is equipped with a wrist-mounted RGB-D camera (RealSense D435) and a parallel gripper (2F-85 and 2F-140). 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. 9.

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 [42] with the Segment Anything Model 2 [45] for object detection and segmentation.

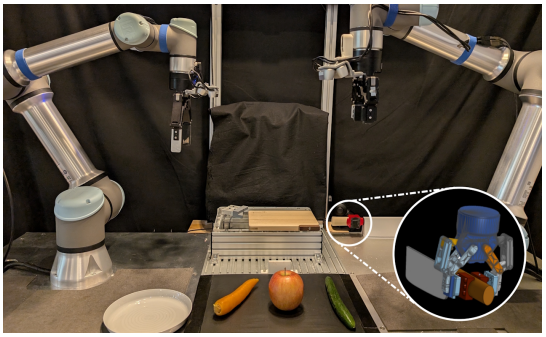


Fig. 9. **Overview of our robotic system.** Dual arm robotic system for real-world validation on a cooking domain.

Pre-defined Skill Library: In ViLaIn-TAMP, we build a library of skills for subtasks involving complex, contact-rich interactions. For instance, in the Slice Food task, slicing actions rely on Reinforcement Learning (RL) with compliance control. Prior work [39] showed this approach adapts to unseen objects and can capture an object's force profile with a single slice. As formalized in Sect. IV-B, each skill is a black-box operator with symbolic pre/post conditions and object-centric start/end poses; the RL policy is short-horizon, and execution halts if the ending pose is not reached. The library can be further extended by training additional skills in the same way.

Real-World Execution: 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.

Real-world validation confirmed the overall feasibility of ViLaIn-TAMP on a physical dual-arm system, though two main failure modes were observed. The most frequent involved object displacement during slicing, leaving objects in poses that the subsequent grasping actions could not recover from. A secondary challenge was grasping deformable objects: vegetable slices can be easily squeezed and become slippery, making open-loop grasping unreliable.

VI. Conclusion

This work introduced ViLaIn-TAMP, a hybrid planning framework that integrates VLMs with symbolic and geometric planning to enable robust, interpretable, and autonomous bimanual robot manipulation in real-world environments. Using ViLaIn to translate multimodal input into PDDL problems and a TAMP system to find action plans, our framework systematically verifies the logical and physical feasibility of the generated plans. Across five challenging bimanual cooking tasks, ViLaIn-TAMP outperformed VLM-as-a-planner baselines with increased interpretability, especially as task complexity increases. The corrective planning module, which iteratively refines plans based on grounded failure feedback from our custom MTC implementation, proved essential for robustness. In future work, ViLaIn-TAMP can be extended with automatic PDDL domain generation [38] and visual failure reasoning to further reduce manual effort and improve error recovery.

References

[1] OpenAI, "GPT-4o system card," *arXiv*, 2024.
[2] A. Grattafiori *et al.*, "The Llama 3 herd of models," *arXiv*, 2024.
[3] G. Team, "Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context," *arXiv*, 2024.
[4] W. Huang *et al.*, "Inner monologue: Embodied reasoning through planning with language models," in *CoRL*, 2023, pp. 1769–1782.
[5] I. Singh *et al.*, "ProgPrompt: Generating situated robot task plans using large language models," in *ICRA*, 2023, pp. 11 523–11 530.
[6] T. Silver *et al.*, "Generalized planning in PDDL domains with pretrained large language models," in *AAAI*, 2024, pp. 20 256–20 264.
[7] A. Mei *et al.*, "ReplanVLM: Replanning robotic tasks with visual language models," *RA-L*, vol. 9, no. 11, pp. 10 201–10 208, 2024.
[8] S. Wang *et al.*, "LLM3: Large language model-based task and motion planning with motion failure reasoning," in *IROS*, 2024, pp. 12 086–12 092.
[9] Z. Yang *et al.*, "Guiding long-horizon task and motion planning with vision language models," in *ICRA*, 2025, pp. 16 847–16 853.
[10] K. Shirai *et al.*, "Vision-language interpreter for robot task planning," in *ICRA*, 2024, pp. 2051–2058.
[11] M. Fox and D. Long, "PDDL2.1: An extension to PDDL for expressing temporal planning domains," *JAIR*, vol. 20, pp. 61–124, 2003.
[12] Y. Zhu *et al.*, "Vima: General robot manipulation with multimodal prompts," in *ICLR*, 2023.
[13] D. Coleman *et al.*, "Reducing the barrier to entry of complex robotic software: a MoveIt! case study," *ArXiV*, 2014.
[14] P. M. Fresnillo *et al.*, "Extending the motion planning framework—MoveIt with advanced manipulation functions for industrial applications," *Robotics and Computer-Integrated Manufacturing*, vol. 83, p. 102559, 2023.
[15] M. Görner *et al.*, "MoveIt! Task Constructor for Task-Level Motion Planning," in *ICRA*, 2019, pp. 190–196.
[16] M. Helmert, "The fast downward planning system," *JAIR*, vol. 26, pp. 191–246, 2006.
[17] OpenAI, "GPT-4 technical report," *arXiv*, 2024.
[18] K. Kawaharazuka *et al.*, "Real-world robot applications of foundation models: a review," *Advanced Robotics*, vol. 38, pp. 1232–1254, 2024.
[19] S. S. Raman *et al.*, "Planning with large language models via corrective re-prompting," in *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.
[20] K. Lin *et al.*, "Text2Motion: From natural language instructions to feasible plans," *Autonomous Robots*, vol. 47, pp. 1345–1365, 2023.
[21] B. Liu *et al.*, "LLM+P: Empowering large language models with optimal planning proficiency," *ArXiV*, 2023.
[22] Y. Xie *et al.*, "Translating natural language to planning goals with large-language models," *ArXiv*, 2023.
[23] C. Huang and L. Zhang, "On the limit of language models as planning formalizers," in *ACL*, 2025.
[24] C. R. Garrett *et al.*, "PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning," in *ICAPS*, vol. 30, 2020, pp. 440–448.
[25] J. Siburian *et al.*, "Practical task and motion planning for robotic food preparation," in *SII*, 2025, pp. 1229–1234.
[26] A. Curtis *et al.*, "Long-horizon manipulation of unknown objects via task and motion planning with estimated affordances," in *ICRA*, 2022, pp. 1940–1946.
[27] Y. Ding *et al.*, "Task and motion planning with large language models for object rearrangement," in *IROS*, 2023, pp. 2086–2092.
[28] K. Rana *et al.*, "Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning," in *CoRL*, vol. 229, 2023, pp. 23–72.
[29] Y. Chen *et al.*, "AutoTAMP: Autoregressive task and motion planning with LLMs as translators and checkers," in *ICRA*, 2024, pp. 6695–6702.
[30] W. Guo *et al.*, "Castl: Constraints as specifications through llm translation for long-horizon task and motion planning," in *ICRA*, 2025, pp. 11 957–11 964.
[31] N. T. Dantam *et al.*, "Incremental task and motion planning: A constraint-based approach," in *RSS*, 2016.
[32] A. O'Neill *et al.*, "Open X-Embodiment: Robotic learning datasets and RT-X models," in *ICRA*, 2024, pp. 6892–6903.
[33] S. S. Raman *et al.*, "Cape: Corrective actions from precondition errors using large language models," in *ICRA*, 2024, pp. 14 070–14 077.
[34] J. Duan *et al.*, "AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation," in *ICLR*, 2025.
[35] C. R. Garrett *et al.*, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, no. 1, pp. 265–293, 2021.
[36] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez, "Learning symbolic operators for task and motion planning," in *IROS*, 2021, pp. 3182–3189.
[37] P. Smirnov *et al.*, "Generating consistent pddl domains with large language models," *ArXiV*, 2024.
[38] J. Oswald *et al.*, "Large language models as planning domain generators," in *ICAPS*, vol. 34, 2024, pp. 423–431.
[39] C. C. Beltran-Hernandez *et al.*, "SliceIt! - a dual simulator framework for learning robot food slicing," in *ICRA*, 2024, pp. 4296–4302.
[40] G. Liu *et al.*, "Optimistic reinforcement learning-based skill insertions for task and motion planning," *RA-L*, 2024.
[41] B. Hedegaard *et al.*, "Beyond task and motion planning: Hierarchical robot planning with general-purpose skills," *arXiv*, 2024.
[42] S. Bai *et al.*, "Qwen2.5-VL technical report," *ArXiV*, 2025.
[43] M. Abdin *et al.*, "Phi-4 technical report," *arXiv*, 2024.
[44] B. Hui *et al.*, "Qwen2.5-coder technical report," *arXiv*, 2024.
[45] N. Ravi *et al.*, "SAM 2: Segment anything in images and videos," *ArXiV*, 2024.