PLanAR: Planning-Language-Grounded Agentic Reasoning for Robot Manipulation

Pengyuan Guo[1,*], Zhonghao Mai[1,*], Zhengtong Xu[1,*], Kaidi Zhang[1], Quan Khanh Luu[1]
Heng Zhang[2], Zichen Miao[1], Arash Ajoudani[2], Zachary Kingston[1], Qiang Qiu[1], Yu She[1]

[1]Purdue University [2]Istituto Italiano di Tecnologia
[*]Equal contribution
https://planar-robot.github.io/

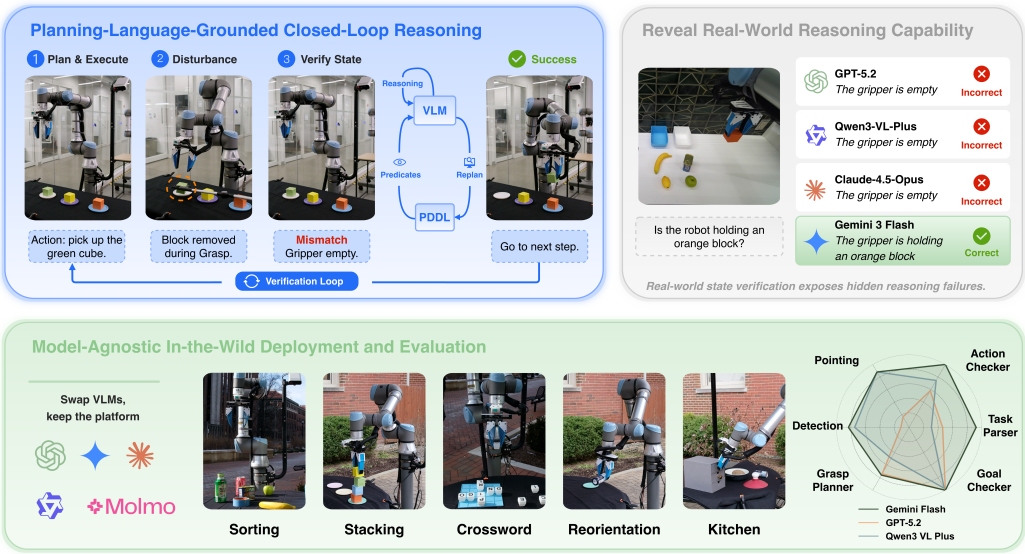


Figure 1: PLanAR is a planning-language-grounded robot agent framework that enables vision-language models to perform structured agentic reasoning for long-horizon manipulation in unstructured real-world environments. With onboard perception and open-vocabulary prompts, PLanAR iteratively performs task planning, execution, online verification, and replanning.

Abstract: Recent advances in vision-language models (VLMs) have enabled increasing progress in real-world robot manipulation. However, long-horizon manipulation in unstructured environments requires VLMs to reason about changing scene states, action constraints, and execution outcomes, which remains difficult with natural language reasoning alone. We present PLanAR, a planning-language-grounded robot agent framework for open-vocabulary, long-horizon manipulation. PLanAR uses a planning-language interface to define the VLM reasoning space: object predicates represent scene states, action schemas specify robot skills with preconditions and effects, and symbolic plans provide executable intermediate representations. This interface enables stepwise verification: after each action, PLanAR uses onboard observations to check whether the expected symbolic effects have been achieved, allowing the VLM-based agent to update task states, detect failures, and replan when execution deviates from expectation. Across robot embodiments, VLM backends, and tasks including stacking, crossword solving, and long-horizon kitchen workflows, PLanAR demonstrates strong real-world capability while revealing key limitations of current VLMs in embodied reasoning.

Keywords: Robot Manipulation, Vision-Language Models, Agentic Reasoning

1 Introduction

Recent progress in large vision-language models (VLMs) has enabled strong open-vocabulary visual understanding [1, 2, 3]. These capabilities have motivated growing interest in using VLMs for real-robot manipulation [4, 5, 2]. However, long-horizon manipulation in unstructured environments requires more than perception and one-step decision making. A robot must reason about changing scene states, action constraints, execution outcomes, and recovery as the world evolves [5, 6].

Existing methods are often limited by scene diversity, task complexity, or execution robustness: some operate in constrained settings with simplified object states [7, 8], others ground instructions in geometric representations without long-horizon symbolic reasoning [9, 10, 11, 12], and many rely on one-shot code generation or open-loop planning and execution without explicit verification and replanning [13, 14]. In parallel, vision-language-action models attempt to unify perception, reasoning, and control through end-to-end action prediction [15, 4, 16, 17, 18], but strong performance often requires task-specific fine-tuning, which can reduce the open-world generalization inherited from pretrained models [19, 20, 21, 22, 23].

A key challenge in using VLMs for manipulation is that natural language reasoning alone does not provide a reliable interface for long-horizon robot execution. Without an explicit representation of what states are relevant, which skills are admissible, and what each skill is expected to change, the agent can lose consistency across steps, hallucinate object states, or fail to recognize when replanning is needed. These issues compound under real-world occlusion, disturbances, and partial observability.

Planning languages provide a natural way to make long-horizon robot reasoning explicit and verifiable [24, 25, 26]. In classical planning, the Planning Domain Definition Language (PDDL) defines a domain through object predicates, action schemas, preconditions, and effects [27, 28]. For VLM-based robot agents, this representation can serve as an explicit reasoning space: predicates make scene states explicit, action schemas constrain the agent to admissible robot skills, and preconditions and effects expose the causal structure of each action. Moreover, expected symbolic effects provide a direct mechanism for closed-loop verification. After executing an action, the agent can verify from onboard observations whether the intended state transition has been achieved and replan when needed. In this way, planning-language grounding supports not only task decomposition, but also online verification, failure recovery, and disturbance handling.

In this paper, we present **PLanAR**, planning-language-grounded agentic reasoning for robot manipulation. Our contributions are threefold.

1. **Planning-Language-Grounded Closed-Loop Reasoning.** PLanAR uses explicit object predicates, action schemas, preconditions, and effects. Since each action has expected symbolic effects, PLanAR can verify outcomes, update state, detect failures, and replan when necessary.

2. **Compositional Agentic Pipeline.** We instantiate this formulation as a compositional robot agent pipeline that decomposes long-horizon manipulation into perception, symbolic state estimation, task planning, skill execution, outcome verification, and replanning. This modular design connects VLM reasoning with executable robot skills while keeping each stage inspectable and replaceable, enabling systematic analysis of where embodied reasoning succeeds or fails.

3. **Generalist Real-World Deployment.** We build PLanAR as a real-world robot agent platform for in-the-wild manipulation. PLanAR supports different VLM backends and robot embodiments, enabling model-agnostic, cross-embodiment deployment on long-horizon tasks such as spatially constrained stacking, crossword solving, and kitchen workflows.

2 Related Work

2.1 VLM-based Robot Manipulation

Recent progress in VLMs has motivated their use as planning and reasoning modules for robot manipulation. A common paradigm is to condition a VLM on visual observations and language

instructions to generate plans [29, 13, 30, 14, 31] or programs [9, 10, 32, 33], which are then translated into executable skill calls handled by separate perception and control modules. While promising, these systems are often constrained by curated scenes [7, 9], simplified object states [13, 33, 8, 34, 35], and short-horizon tasks [14, 29, 10].

Some work grounds language instructions in spatial or geometric representations, such as object-centric constraints, affordance maps, keypoints, or spatial value functions [9, 10, 11, 12, 36, 37, 38]. These methods are effective for geometry-dominated manipulation, but place less emphasis on semantic long-horizon reasoning over evolving symbolic states and execution outcomes.

Closed-loop execution remains challenging. Many VLM-based systems rely on one-shot code generation, open-loop planning, or limited feedback after execution [13, 14]. Without explicit expected effects for each action, it is difficult to verify success, update task states, or decide when to replan. Existing task-specific checks or corrective behaviors [32, 31] are often heuristic rather than grounded in a unified reasoning representation, making long-horizon execution brittle under object movement, occlusion, grasp failures, disturbances, and background changes.

PLanAR addresses these limitations by grounding VLM reasoning in a planning-language interface for manipulation. Object predicates, action schemas, preconditions, and effects define an explicit reasoning space, while expected symbolic effects provide natural verification targets. This enables closed-loop agentic reasoning: the agent checks execution outcomes from onboard observations, and replans when the scene deviates from expectation.

2.2 Embodied Benchmarks for VLMs

Many embodied benchmarks for VLMs use question-answering or offline interfaces to evaluate visual and spatial language reasoning without executing actions in the world [5, 39, 40, 41, 6, 42]. While useful for controlled diagnosis, these settings cannot test whether a model can maintain grounding under action-induced scene changes, verify execution outcomes, recover from failures, or handle error accumulation over long horizons.

Simulation-based benchmarks provide scale through standardized task suites [43, 44, 45, 46, 47, 48, 21, 22, 49]. However, simulated environments often simplify physical dynamics, object states, perception noise, and action execution through cleaner scenes or abstracted primitives. As a result, simulation success may overestimate robustness and does not always translate to reliable real-world manipulation.

PLanAR provides real-world, closed-loop evaluation of VLM agents on physical robot tasks, revealing embodied reasoning limitations under clutter, occlusion, lighting variation, object movement, and scene changes that offline and simulation benchmarks often miss.

3 PLanAR Framework

As shown in Fig. 2, PLanAR connects natural-language task instructions, onboard observations, symbolic planning, robot skill execution, and closed-loop verification within a unified pipeline. To bridge unstructured language instructions and executable robot logic, PLanAR grounds VLM reasoning in a PDDL interface, building on LLM-guided symbolic planning formulations [24, 7] and extending them to closed-loop verification and replanning. The PDDL domain defines the agent's symbolic reasoning space through object types, predicates, action schemas, preconditions, and effects, such as whether an object is on a support surface or held by the gripper. Given a generated PDDL problem and the predefined domain, PLanAR invokes Fast Downward [28] to compute a sequence of high-level symbolic actions that satisfy the task goal. The resulting plan provides an executable symbolic interface between VLM reasoning and robot control. Implementation details are provided in Appendix A.

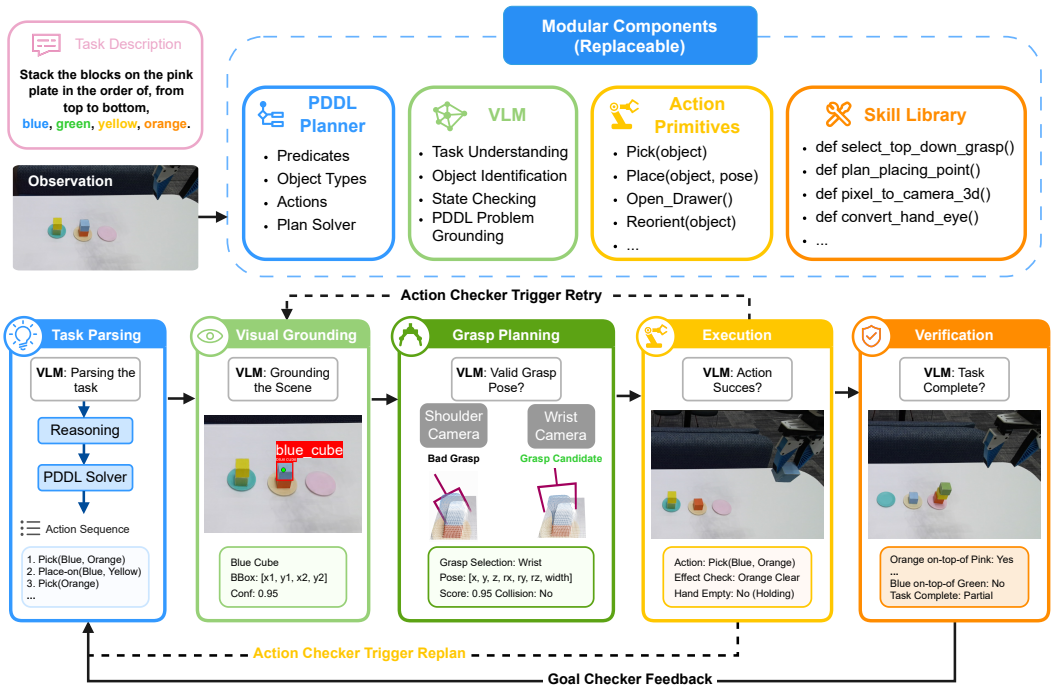


Figure 2: Pipeline of PLanAR. Given a natural-language instruction and onboard observations, the task parser grounds the instruction into symbolic goals and a PDDL action sequence. Visual grounding and grasp planning then instantiate objects, states, and feasible grasps in the real scene. Action and goal checkers verify states after each step; failed checks trigger per-step retry or replanning with updated observations. All modules are replaceable under the same planning-language interface.

3.1 Symbolic Verification and Replanning

Before each action, the action checker evaluates whether the required preconditions hold and whether the action is feasible in the current scene. After execution, the system re-observes the environment and verifies that the expected effects have taken place, using updated visual input to detect failures such as missed grasps, incorrect object interactions, or unexpected state changes. Upon completion of the full action sequence, a final goal-condition check confirms whether the scene satisfies all specified goal predicates. When a verification step fails, PLanAR triggers replanning: the task parser re-instantiates an updated PDDL problem reflecting the current world state, and the symbolic planner generates a revised action sequence from that state. This closed loop between visual observation, symbolic verification, and plan repair enables robust long-horizon manipulation without assuming perfect execution at each step.

3.2 Converting Symbolic Actions into Robot Skills

To bridge the gap between discrete symbolic plans and continuous physical execution, PLanAR abstracts robot behavior into a library of high-level parameterized action primitives, including pick, place, and open/close drawer. To execute these primitives, PLanAR leverages specialized skill library functions, such as coordinate transformations and placement pose generation, extending the system's capabilities beyond basic pick-and-place to include complex interactions like bottle reorientation, drawer manipulation, and constraint-driven stacking. Visual grounding uses a swappable interface supporting direct VLM prediction or LangSAM [50] segmentation, enabling open-vocabulary detection across models with or without native grounding capability. For grasping actions, PLanAR first grounds the target object from RGB-D observations and generates candidate 6-DoF grasp poses using AnyGrasp [51]. A VLM-based grasp evaluator then checks whether the selected grasp is semantically correct and physically feasible, including whether it targets the intended object and whether it

may cause collision or instability. As shown in Fig. 2, PLanAR first plans grasps from the shoulder-view camera. If the candidate grasp is rejected, the robot switches to the wrist-mounted camera for close-range observation and replans the grasp with more localized visual evidence.

3.3 Implementation Across Robot Embodiments

We instantiate PLanAR on a UR5e platform with two RGB-D views: a fixed shoulder-view camera for global scene grounding and a wrist-mounted camera for close-range grasp verification and replanning. The UR5e is mounted on a movable base for deployment across laboratory settings, general indoor environments, and in-the-wild scenes. It uses a low-cost parallel gripper with fin-ray fingers for repeatable manipulation of diverse everyday objects. Fig. 3 provides an overview of the platform. Because PLanAR exposes a structured interface, models and robot embodiments can be swapped under the same action schema and verification protocol. We also transfer PLanAR to a Franka Panda arm using the same symbolic domain, action schemas, and verification prompts, changing only the low-level execution wrapper.

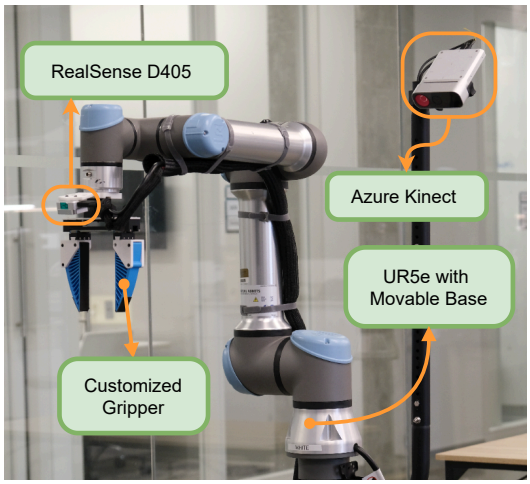


Figure 3: The PLanAR Manipulation Platform: a reproducible, easy-to-deploy platform for real-world in-the-wild manipulation.

4 Experiment

We organize the experiments around four questions. First, how reliably do VLM-based robot agents perform closed-loop manipulation? Second, can PLanAR's modular interface mitigate specific bottlenecks by assigning different VLMs to different modules? Third, how much do planning-language grounding and execution-time verification contribute to robustness? Finally, can PLanAR transfer across robot embodiments and compare favorably against VLA-based baselines?

We validate PLanAR on two real robot platforms. On the UR5e, we evaluate five tasks, Sorting, Stacking, Crossword, Reorientation, and Kitchen, across laboratory, indoor, and outdoor settings, as shown in Fig. 21. On Franka Panda, we evaluate Sorting and Stacking for cross-embodiment comparison, including direct VLM planning and fine-tuned VLA baselines. These tasks test open-vocabulary grounding, spatial reasoning, symbolic state tracking, and recovery under real-world uncertainty.

4.1 Diagnosing VLM Bottlenecks in Closed-Loop Manipulation

We first evaluate eight single-VLM pipelines on the sorting task, using 20 real-robot trials across six object setups spanning fruits, blocks, toys, and tools. Fig. 4 summarizes dominant failure modes; full rollout statistics are reported in Appendix Table 4.

Gemini Flash is the most reliable single-backend pipeline, combining low latency with stable visual grounding. Gemini Pro is strong in abstract reasoning but less reliable in visual grounding; replacing its detector with LangSAM improves performance.

Models such as Qwen-VL-Max and GPT-5.2 achieve near-zero success primarily due to unreliable execution-state verification. A common failure mode is hallucination during action checking. For example, the model may claim the object is not in the gripper after a successful pick.

To attribute these rollout failures to concrete sources, we further evaluate five modules in the PLanAR pipeline: task parser, object detector, action checker, grasp planner, and goal checker, with full results

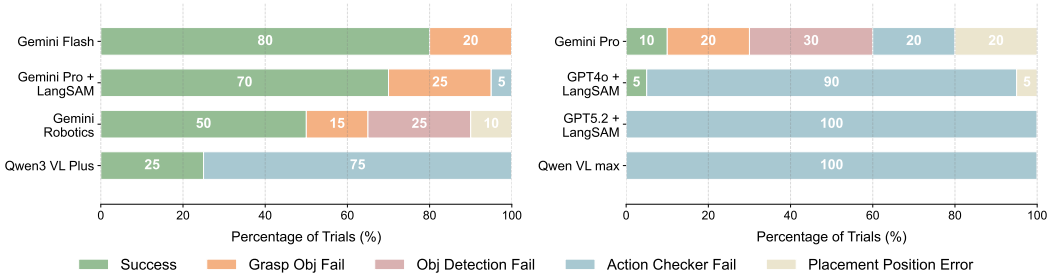


Figure 4: Failure mode breakdown for single-VLM pipelines on the sorting task. Results show which pipeline components dominate failures across different VLMs.

Table 1: Module evaluation of VLMs for PLanAR. SR denotes success rate; "–" indicates the VLM cannot generate usable results for the module.

Model		Task Parser		Action Checker		Object Detector			Grasp Planner		Goal Checker	
		SR (%)	Time	SR (%)	Time	Pointing SR (%)	Bbox IoU (%)	Time	SR (%)	Time	SR (%)	Time
Gemini	Gemini 3 Pro	**92.3**	70.5	98.4	13.3	66.7	53.7	45.7	**77.8**	26.8	**100.0**	8.4
	Gemini 3 Flash	**92.3**	45.2	**100.0**	**5.3**	**88.1**	**83.4**	9.5	74.1	9.1	**100.0**	15.6
	Gemini Robotics	75.0	29.6	95.1	10.1	57.1	40.4	5.8	55.0	10.3	**100.0**	13.5
Qwen	Qwen3-VL-Plus	30.8	40.2	74.6	11.4	**88.1**	74.8	5.1	66.7	10.0	96.9	12.1
	Qwen-VL-Max	17.3	36.3	64.8	24.2	81.0	60.3	5.0	59.2	9.7	84.4	11.2
Others	GPT-5.2	46.2	31.8	58.2	8.6	16.6	11.2	**3.2**	75.9	8.0	**100.0**	9.1
	GPT-4o	48.1	**16.0**	68.9	7.0	14.3	8.0	3.9	44.4	**4.4**	93.8	**5.1**
	Claude-4.5-Opus	78.9	17.0	64.8	7.1	9.6	6.1	4.4	70.4	8.2	93.8	6.6
	Claude-4.5-Sonnet	40.4	17.2	40.4	11.9	61.9	43.9	4.1	54.7	9.6	78.1	10.0
Local	Qwen2.5-VL-7B	–	–	25.8	8.1	73.8	51.0	6.6	–	–	50.0	9.2
	Molmo-7B	–	–	33.9	19.0	76.2	27.0	3.6	–	–	43.8	18.1

reported in Table 1. Task parsing is generally not the main bottleneck for cloud-hosted VLMs. Instead, failures are dominated by grounding and verification, whose errors compound because they are repeatedly invoked during execution. For example, in our sorting task, placing three objects triggers six calls to the action checker. As a result, even with 90% accuracy on static VQA, the probability of completing a full trial is only $0.9^6 \approx 0.53$. More critically, at 50% accuracy, success collapses to $0.5^6 \approx 1.5\%$. This compounding effect explains why models with seemingly moderate verification errors in isolation can fail catastrophically under closed-loop execution.

4.2 Modular VLM Composition

Motivated by the module-level evaluation, we next ask whether PLanAR's modular interface can mitigate specific bottlenecks through model assignment. We construct a compositional pipeline that assigns different VLM backends to different modules: Gemini Flash for the task parser and grasp planner, Qwen3-VL-Plus for the object detector, and Claude Opus for the goal checker. These models are selected based on their competitive module-level performance. We compare this compositional pipeline against a strong single-backend Gemini Flash baseline across laboratory and kitchen settings, with results reported in Fig. 5.

For this experiment, we report a task progress score computed from a structured rubric in Appendix B.1. This metric captures partial completion and is more informative than binary success. To reduce rollout latency in longer tasks, we disable action-level verification and retain only the goal checker. We analyze the cost–benefit of dense action verification separately in Appendix B.4.

As shown in Fig. 5, the compositional pipeline does not consistently outperform the Gemini Flash baseline. Both pipelines show a robustness gap when moving from controlled laboratory trials to kitchen settings, with the largest drops in the Crossword and Kitchen tasks, where precise placement and sustained object-identity tracking are critical.

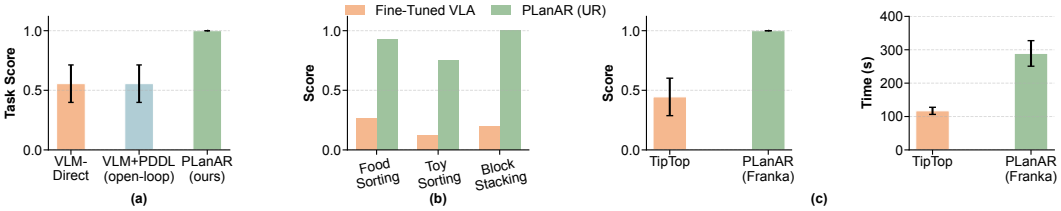


Figure 6: Ablations and baseline comparisons. **(a)** Comparison of VLM-Direct, VLM+PDDL (Open-Loop), and PLanAR on fruit sorting with human disturbance. **(b)** Comparison between PLanAR and a fine-tuned $\pi_{0.5}$ VLA baseline. **(c)** Cross-embodiment evaluation on the Franka robot and comparison with TiPToP on fruit sorting with human disturbance.

This suggests that model composition is not universally better than a strong single-VLM pipeline; its benefit depends on the task and scene. For example, replacing the grounding module with Qwen3-VL-Plus improves stacking, where localization and precise placement are important. In the kitchen setting, however, the same substitution exposes Qwen's spatial-grounding weakness, such as confusing the middle drawer with the top drawer, and performs worse than Gemini Flash. For the reorientation task, we use a task-specific Gemini 3 Pro variant for pose-conditioned reasoning, as most models struggle to jointly reason about the grasped bottle's orientation and its spatial relationship to the table. Overall, we use composition as a targeted strategy to exploit model complementarity when the bottleneck is identifiable, rather than as a universally superior design.

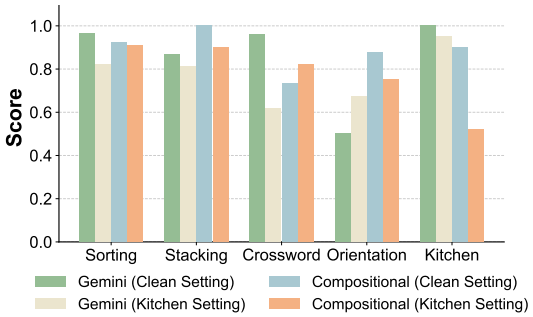


Figure 5: Performance comparison across environments. We compare a Gemini Flash baseline with a compositional pipeline across five manipulation tasks. Performance is measured using a task progress score that quantifies partial completion.

4.3 Ablation: Planning-Language Grounding and Closed-Loop Verification

We ablate the roles of planning-language grounding and execution-time verification on our Franka Panda adaptation of PLanAR. All configurations use the same VLM backend, perception stack, robot platform, and skill library: **VLM-Direct**, which directly predicts an action sequence; **VLM+PDDL (Open-Loop)**, which executes a symbolic plan without verification; and **PLanAR**, which adds step-wise precondition/effect verification and replanning.

We evaluate all configurations on fruit sorting, where the robot must place three fruits into a target bowl. To test robustness under execution uncertainty, a human removes the second grasped fruit and returns it to the table, creating a state mismatch similar to misgrasps, object slips, or failed placements.

As shown in Fig. 6(a), VLM-Direct and VLM+PDDL (Open-Loop) exhibit similar performance: both continue execution without detecting that the second grasp no longer holds. VLM-Direct lacks an explicit intermediate state representation for checking execution outcomes; VLM+PDDL provides a symbolic plan, but without execution-time verification it still proceeds under an invalid state assumption. In contrast, PLanAR detects that the expected grasp effect failed, updates the symbolic state, replans, and completes the task. This result shows that planning-language grounding structures action generation, but robustness comes from verifying symbolic effects during execution.

4.4 Comparison with VLA Baselines and Cross-Embodiment Generalization

Finally, we evaluate PLanAR against VLA-based baselines and examine whether its planning-language-grounded design transfers across robot embodiments. We first compare PLanAR with

a fine-tuned $\pi_{0.5}$ baseline [4]. The $\pi_{0.5}$ policy is fine-tuned with 40 demonstrations for sorting and 30 demonstrations for stacking. We then compare PLanAR with TiPToP [14] on a Franka Panda. Across these experiments, PLanAR uses the same high-level task interface, perception stack, planning-language grounding, and verification logic, while only adapting the low-level execution wrappers for each robot. Prompts, task definitions, and evaluation metrics follow the protocol in Sec. 4.2; additional training and implementation details are provided in the Appendix B.6.

As shown in Fig. 6(b), PLanAR outperforms the fine-tuned VLA baseline in our evaluated sorting and stacking tasks. Although the VLA can learn useful visuomotor behaviors from task-specific demonstrations, it struggles with fine-grained semantic constraints and precise object placement. For example, in sorting, the policy may grasp a food item when instructed to pick a toy; in stacking, failures often result from imprecise grasps, unstable placements, or the lack of recovery after intermediate errors. In contrast, PLanAR explicitly decomposes the task, tracks symbolic state, and verifies execution outcomes, enabling more reliable recovery during long-horizon manipulation.

Fig. 6(c) further compares PLanAR with TiPToP on the Franka Panda using the same disturbed fruit-sorting setup as Sec. 4.3, as well as the block stacking task. In fruit sorting, TiPToP struggles under disturbance: after the second fruit is removed from the gripper and returned to the table, it does not recover from the resulting state mismatch. Additional failures also arise from incorrect grasps produced by Contact-GraspNet [52]. On block stacking, TiPToP is unable to complete the task in our setup because cuTAMP [53] does not generate feasible motion paths for the required stacking actions. In contrast, PLanAR achieves higher scores on both tasks by explicitly verifying execution outcomes, updating the task state, and replanning after disturbances or failed grasps. These results suggest that closed-loop reasoning and verification are particularly useful for tasks involving spatial constraints, sequential dependencies, and execution uncertainty. This robustness, however, comes at the cost of longer runtime, since PLanAR invokes additional verification and replanning steps after detected failures. Additional details are shown in the Appendix B.7.

5 Conclusion & Limitations

In this work, we presented PLanAR, a planning-language-grounded robot agent framework for real-world manipulation. PLanAR integrates VLM-based task understanding, PDDL-based planning, action-level verification, and closed-loop replanning into a model-agnostic pipeline. PLanAR provides a unified evaluation and deployment interface that plugs VLMs into different modules and exposes their strengths and weaknesses under a shared real-robot protocol. Results across lab, indoor, and in-the-wild scenes show that planning-language grounding structures action generation, while online verification detects failures, corrects state mismatches, and supports recovery. The same reasoning and verification logic transfers across robot embodiments by adapting only the low-level primitive interface. These findings suggest that robust robot agents should optimize for closed-loop consistency over richer open-loop reasoning.

Despite these promises, several limitations remain. First, PLanAR currently relies on predefined PDDL domains. Building a reliable domain requires iterative effort to refine predicates, action schemas, and effects. Second, verification remains latency-sensitive: dense VLM-based checking improves recovery, but replanning and grounding may require frequent model calls during execution. Future systems could reduce this overhead through asynchronous planning and grounding. Finally, PLanAR uses a compact set of robot primitives. Although these primitives support a diverse set of long-horizon tasks, they limit the range of skills that can be evaluated. Extending the primitive interface to more complex and freeform skills, including manipulation behaviors that cannot be reduced to grasp-and-place trajectories, is an important direction for scaling PLanAR.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] G. Comanici, E. Bieber, M. Schaekermann, I. Pasupat, N. Sachdeva, I. Dhillon, M. Blistein, O. Ram, D. Zhang, E. Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

[3] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[4] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, et al. pi0.5: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[5] 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. *arXiv preprint arXiv:2503.20020*, 2025.

[6] G. R. Team, A. Abdolmaleki, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, A. Balakrishna, N. Batchelor, A. Bewley, J. Bingham, et al. Gemini robotics 1.5: Pushing the frontier of generalist robots with advanced embodied reasoning, thinking, and motion transfer. *arXiv preprint arXiv:2510.03342*, 2025.

[7] X. Zhang, Z. Altaweel, Y. Hayamizu, Y. Ding, S. Amiri, H. Yang, A. Kaminski, C. Esselink, and S. Zhang. Dkprompt: Domain knowledge prompting vision-language models for open-world planning. *arXiv preprint arXiv:2406.17659*, 2024.

[8] X. Zhang, Y. Ding, Y. Hayamizu, Z. Altaweel, Y. Zhu, Y. Zhu, P. Stone, C. Paxton, and S. Zhang. Llm-grop: Visually grounded robot task and motion planning with large language models. *The International Journal of Robotics Research*, 2025.

[9] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[10] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei. Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation. *arXiv preprint arXiv:2409.01652*, 2024.

[11] J. Wu, R. Antonova, A. Kan, M. Lepert, A. Zeng, S. Song, J. Bohg, S. Rusinkiewicz, and T. Funkhouser. Tidybot: Personalized robot assistance with large language models. *Autonomous Robots*, 47(8):1087–1102, 2023.

[12] P. Liu, Y. Orru, J. Vakil, C. Paxton, N. M. M. Shafiullah, and L. Pinto. Ok-robot: What really matters in integrating open-knowledge models for robotics. *arXiv preprint arXiv:2401.12202*, 2024.

[13] H. Ye, Y. Xiao, C. Lu, and P. Cai. Pretraining a unified pddl domain from real-world demonstrations for generalizable robot task planning. *arXiv preprint arXiv:2507.21545*, 2025.

[14] W. Shen, N. Kumar, S. Chintalapudi, J. Wang, C. Watson, E. Hu, J. Cao, D. Jayaraman, L. P. Kaelbling, and T. Lozano-Pérez. Tiptop: A modular open-vocabulary planning system for robotic manipulation. *arXiv preprint arXiv:2603.09971*, 2026.

[15] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[16] H. Fang, J. Duan, D. Clay, S. Wang, S. Liu, W. Huang, X. Fan, W.-C. Tsai, S. Chen, Y. R. Wang, et al. Molmoact2: Action reasoning models for real-world deployment. *arXiv preprint arXiv:2605.02881*, 2026.

[17] L. Zha, A. J. Hancock, M. Zhang, T. Yin, Y. Huang, D. Shah, A. Z. Ren, and A. Majumdar. Lap: Language-action pre-training enables zero-shot cross-embodiment transfer. *arXiv preprint arXiv:2602.10556*, 2026.

[18] P. Intelligence, B. Ai, A. Amin, R. Aniceto, A. Balakrishna, G. Balke, K. Black, G. Bokinsky, S. Cao, T. Charbonnier, et al. pi0.7: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

[19] S. Fei, S. Wang, J. Shi, Z. Dai, J. Cai, P. Qian, L. Ji, X. He, S. Zhang, Z. Fei, et al. Libero-plus: In-depth robustness analysis of vision-language-action models. *arXiv preprint arXiv:2510.13626*, 2025.

[20] X. Zhou, Y. Xu, G. Tie, Y. Chen, G. Zhang, D. Chu, P. Zhou, and L. Sun. Libero-pro: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025.

[21] Y. Kim, W. Pumacay, O. Rayyan, M. Argus, W. Han, E. VanderBilt, J. Salvador, A. Deshpande, R. Hendrix, S. Jauhri, et al. Molmospaces: A large-scale open ecosystem for robot navigation and manipulation. *arXiv preprint arXiv:2602.11337*, 2026.

[22] X. Yang, R. Dagli, A. Zook, H. Hadfield, A. Goyal, S. Birchfield, F. Ramos, and J. Tremblay. Robolab: A high-fidelity simulation benchmark for analysis of task generalist policies. *arXiv preprint arXiv:2604.09860*, 2026.

[23] L. X. Shi, B. Ichter, M. Equi, L. Ke, K. Pertsch, Q. Vuong, J. Tanner, A. Walling, H. Wang, N. Fusai, et al. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *arXiv preprint arXiv:2502.19417*, 2025.

[24] 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.

[25] 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 international conference on automated planning and scheduling*, volume 30, pages 440–448, 2020.

[26] M. Kwon, Y. Kim, and Y. J. Kim. Fast and accurate task planning using neuro-symbolic language models and multi-level goal decomposition. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16195–16201. IEEE, 2025.

[27] 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.

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

[29] F. Liu, K. Fang, P. Abbeel, and S. Levine. Moka: Open-world robotic manipulation through mark-based visual prompting. *arXiv preprint arXiv:2403.03174*, 2024.

[30] H.-T. L. Chiang, Z. Xu, Z. Fu, M. G. Jacob, T. Zhang, T.-W. E. Lee, W. Yu, C. Schenck, D. Rendleman, D. Shah, et al. Mobility vla: Multimodal instruction navigation with long-context vlms and topological graphs. *arXiv preprint arXiv:2407.07775*, 2024.

[31] J. Duan, W. Yuan, W. Pumacay, Y. R. Wang, K. Ehsani, D. Fox, and R. Krishna. Manipulate-anything: Automating real-world robots using vision-language models. *arXiv preprint arXiv:2406.18915*, 2024.

[32] M. Fu, J. Yu, K. El-Refai, E. Kou, H. Xue, H. Huang, W. Xiao, G. Wang, F.-F. Li, G. Shi, J. Wu, S. Sastry, Y. Zhu, K. Goldberg, and L. J. Fan. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation, 2026. URL https://arxiv.org/abs/2603.22435.

[33] Y. Jin, D. Li, J. Shi, P. Hao, F. Sun, J. Zhang, B. Fang, et al. Robotgpt: Robot manipulation learning from chatgpt. *IEEE Robotics and Automation Letters*, 9(3):2543–2550, 2024.

[34] R. Gong, X. Zhang, J. Shang, M. V. Minniti, J. Patel, V. Pepe, R. Yan, A. Gundogdu, I. Kapelyukh, A. Abbas, et al. Anytask: an automated task and data generation framework for advancing sim-to-real policy learning. *arXiv preprint arXiv:2512.17853*, 2025.

[35] A. Athalye, N. Kumar, T. Silver, Y. Liang, J. Wang, T. Lozano-Pérez, and L. P. Kaelbling. From pixels to predicates: Learning symbolic world models via pretrained vision-language models, 2026. URL https://arxiv.org/abs/2501.00296.

[36] K. Fang, Y. Chen, X. Zhu, F. Niroui, L. Sun, and J. Wang. Saga: Open-world mobile manipulation via structured affordance grounding. *arXiv e-prints*, pages arXiv–2512, 2025.

[37] S. Patel, X. Yin, W. Huang, S. Garg, H. Nayyeri, L. Fei-Fei, S. Lazebnik, and Y. Li. A real-to-sim-to-real approach to robotic manipulation with vlm-generated iterative keypoint rewards. *arXiv preprint arXiv:2502.08643*, 2025.

[38] N. Kumar, W. Shen, F. Ramos, D. Fox, T. Lozano-Pérez, L. P. Kaelbling, and C. R. Garrett. Open-world task and motion planning via vision-language model generated constraints, 2026. URL https://arxiv.org/abs/2411.08253.

[39] C. H. Song, V. Blukis, J. Tremblay, S. Tyree, Y. Su, and S. Birchfield. Robospatial: Teaching spatial understanding to 2d and 3d vision-language models for robotics. *CoRR*, 2024.

[40] A. Majumdar, A. Ajay, X. Zhang, P. Putta, S. Yenamandra, M. Henaff, S. Silwal, P. Mcvay, O. Maksymets, S. Arnaud, et al. Openeqa: Embodied question answering in the era of foundation models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16488–16498, 2024.

[41] A.-C. Cheng, H. Yin, Y. Fu, Q. Guo, R. Yang, J. Kautz, X. Wang, and S. Liu. Spatialrgpt: Grounded spatial reasoning in vision-language models. *Advances in Neural Information Processing Systems*, 37:135062–135093, 2024.

[42] Y. Hong, J. Liu, H. Yin, M. Li, L. Guibas, F.-F. Li, J. Wu, and Y. Choi. ESI-Bench: Towards embodied spatial intelligence that closes the perception-action loop. *arXiv preprint*, 2026. URL https://esi-bench.github.io/.

[43] R. Yang, H. Chen, J. Zhang, M. Zhao, C. Qian, K. Wang, Q. Wang, T. V. Koripella, M. Movahedi, M. Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025.

[44] M. Li, S. Zhao, Q. Wang, K. Wang, Y. Zhou, S. Srivastava, C. Gokmen, T. Lee, E. L. Li, R. Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534, 2024.

[45] M. Khanna, R. Ramrakhya, G. Chhablani, S. Yenamandra, T. Gervet, M. Chang, Z. Kira, D. S. Chaplot, D. Batra, and R. Mottaghi. Goat-bench: A benchmark for multi-modal lifelong navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16373–16383, 2024.

[46] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023.

[47] J.-W. Choi, Y. Yoon, H. Ong, J. Kim, and M. Jang. Lota-bench: Benchmarking language-oriented task planners for embodied agents. *arXiv preprint arXiv:2402.08178*, 2024.

[48] X. Liu, T. Zhang, Y. Gu, I. L. Iong, Y. Xu, X. Song, S. Zhang, H. Lai, X. Liu, H. Zhao, et al. Visualagentbench: Towards large multimodal models as visual foundation agents. *arXiv preprint arXiv:2408.06327*, 2024.

[49] Y. Huang, B. Li, V. Saxena, Y. Liang, U. A. Mishra, L. Ji, L. Zha, J. Wu, N. Kumar, S. Scherer, D. Xu, and T. Silver. Kinder: A physical reasoning benchmark for robot learning and planning, 2026. URL https://arxiv.org/abs/2604.25788.

[50] L. Medeiros. Language segment-anything. https://github.com/luca-medeiros/lang-segment-anything, 2023.

[51] H.-S. Fang, M. Gou, C. Wang, and C. Lu. Robust grasping across diverse sensor qualities: The graspnet-1billion dataset. *The International Journal of Robotics Research*, 2023.

[52] M. Sundermeyer, A. Mousavian, R. Triebel, and D. Fox. Contact-graspnet: Efficient 6-dof grasp generation in cluttered scenes. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 13438–13444. IEEE, 2021.

[53] W. Shen, C. Garrett, N. Kumar, A. Goyal, T. Hermans, L. P. Kaelbling, T. Lozano-Pérez, and F. Ramos. Differentiable gpu-parallelized task and motion planning. In *Robotics: Science and Systems*, 2025.

[54] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026, 2023.

[55] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

A Implementation Details

A.1 Implementation Overview

We describe the PLanAR pipeline in Algorithm 1. The system supports multiple action types, such as pick and place. We illustrate the main loop with pick; the remaining actions follow the same structure with minor, action-specific changes. For example, place performs only Detect and does not invoke AnyGrasp.

Algorithm 1 PLanAR Pipeline

Require:
Vision-Language Model $\mathcal{M}$
Task Instruction l
PDDL Domain File D
Text Prompt for Task Parser p_{task}
Text Prompt for Action Checker p_{checker}
Text Prompt for Grasp Planner p_{grasp}
Text Prompt for Goal Checker p_{goal}
Initial Observation s_0^{sh} // "sh" denotes shoulder camera.

// Query $\mathcal{M}$ to parse the task into N actions $\{a_k\}_{k=1}^N$.
// Each action specifies preconditions, target object, and effects.
1: $\{a_k\}_{k=1}^N = \mathcal{M}(p_{\text{task}},\ l,\ D,\ s_0)$
2: **for** action $\{a_k\}_{k=1}^N$ **do**
3: Get observation s_k^{sh}
// Verify precondition.
4: **if not** $\mathcal{M}(p_{\text{checker}}, a_k^{\text{pre}}, s_k^{\text{sh}})$ **then**
5: Go to step 1
6: **end if**
// Detect the target object.
7: $\text{bbox}_k^{\text{sh}} = \text{Detect}(s_k^{\text{sh}}, a_k^{\text{target}})$
// Generate the grasp pose.
8: $\pi_k = \text{AnyGrasp}(s_k^{\text{sh}}, \text{bbox}_k^{\text{sh}}, a_k)$
// Evaluate the generated grasp.
9: **if not** $\mathcal{M}(p_{\text{grasp}}, \pi_k, s_k^{\text{sh}})$ **then**
10: Move end-effector to π_k to get wrist camera view
11: Get observation s_k^{wrist} from wrist camera
12: $\text{bbox}_k^{\text{wrist}} = \text{Detect}(s_k^{\text{wrist}}, a_k^{\text{target}})$
13: $\pi_k = \text{AnyGrasp}(s_k^{\text{wrist}}, \text{bbox}_k^{\text{wrist}}, a_k)$
14: **end if**
15: Execute π_k
16: Get post-action observation s_k^{post}
// Verify effect.
17: **if not** $\mathcal{M}(p_{\text{checker}}, a_k^{\text{effect}}, s_k^{\text{post}})$ **then**
18: Go to step 3
19: **end if**
20: **end for**
// Verify task completion.
21: **if not** $\mathcal{M}(p_{\text{goal}}, s_N^{\text{post}})$ **then**
22: Go to step 1
23: **end if**

A.2 Qualitative Walkthrough and I/O Specifications

We utilize a unified interface for all VLM components. To ensure reproducibility, we provide a concrete walkthrough of the main modules.

Task Parser. To bridge natural-language instructions and symbolic planning, the task parser facilitates the translation from instruction to PDDL. As illustrated in Table 2, it operates as follows:

(i) Input: The model receives the domain definition (Fig. 7), scene observation, and a user instruction (e.g., "stack the cubes...").

(ii) Reasoning: The VLM generates a PDDL problem file (Fig. 8) that defines the initial state and goal conditions based on the visual observation.

(iii) Output: A classical planner, Fast Downward [28], solves this PDDL to produce an executable action sequence (Fig. 9).

PDDL Domain: stacking

```
(define (domain stacking)
   (:requirements :strips :typing)
   (:types cube cloth plate bowl mat paper - object
         purple-cube blue-cube orange-cube yellow-cube green-cube - cube)
   (:predicates
        (on-top-of ?o1 - object ?o2 - object)
        (hand-empty)
        (holding ?o - object)
        (clear ?o - object))

   (:action pick
   :parameters (?obj - object ?underobj - object)
   :precondition (and (hand-empty)
                 (clear ?obj)
                 (on-top-of ?obj ?underobj))
   :effect (and (holding ?obj)
            (clear ?underobj)
            (not (hand-empty))
            (not (on-top-of ?obj ?underobj))
            (not (clear ?obj))))

   (:action place-on-object
   :parameters (?obj - object ?target - object)
   :precondition (and (holding ?obj)
                 (clear ?target))
   :effect (and (on-top-of ?obj ?target)
            (hand-empty)
            (clear ?obj)
            (not (clear ?target))
            (not (holding ?obj)))))
```

Figure 7: The defined PDDL domain of stacking task.

Action Checker. To ensure closed-loop reasoning and robustness, the action checker validates both preconditions and effects. Table 3 demonstrates this capability.

A detailed input example for the action checker is shown in Fig. 10, including representative precondition and effect queries for pick actions under the shoulder camera view. Figs. 11-14 further visualize corresponding JSON outputs from different VLMs, illustrating how the checker reports per-predicate satisfaction and diagnoses failure cases when evaluated effects are not correct.

Grasp Planner. Fig. 15 illustrates the input used to evaluate semantic correctness and physical feasibility: the proposed grasp pose visualized in the local point cloud. We provide ground-truth executability labels (a-e) for better understanding. The corresponding model reasoning and judgments are detailed in Figs. 16-20. The model evaluates whether the pose targets the correct semantic object and is collision-free. Crucially, a `REJECT` decision triggers the system to switch to the wrist camera for a close-up re-evaluation and re-planning.

The input request contains:

- A string describing the multi-stage manipulation task.
- An image of the current table-top environment captured from a shoulder camera.
- A PDDL domain file defining available predicates and actions.
- An example problem in both natural language and PDDL format for few-shot learning.

The output response is a JSON object containing:

- **objects_identified**: A list of objects identified from the scene. Each object has a `name` (PDDL identifier, e.g., "blue-block-1"), `type` (object category), and `description` (natural language description of the object).
- **reasoning**: A string providing brief analysis of task-relevant objects, their types, spatial relationships, and task requirements.
- **updated_domain**: The complete PDDL domain file with new object types added if the task mentions types not in the original domain.
- **problem_pddl**: A complete PDDL problem file containing the `:objects` section listing all identified objects, `:init` section describing the current state, and `:goal` section specifying the desired final state.

Table 2: The high-level reasoning prompt for task parser. It analyzes the scene image and task instruction to generate a structured PDDL representation that can be solved by a classical planner.

The input request contains:

- A PDDL action to be executed (e.g., `(pick apple-1)`).
- The preconditions for the action in PDDL format (e.g., `(and (hand-empty) (clear apple-1))`).
- An image of the current scene captured from a shoulder camera showing the robot gripper and workspace.
- A reference guide explaining PDDL predicates such as `hand-empty`, `holding`, `clear`, `on-table`, and `on-top-of`.

The output response is a JSON object containing:

- **conditions_analysis**: A list of individual condition evaluations. Each entry contains a `condition` (the exact predicate string), `satisfied` (boolean indicating if the condition holds), and `observation` (visual evidence from the image supporting the evaluation).
- **success**: A boolean indicating whether ALL preconditions are satisfied based on logical operators (and/or/not).
- **reasoning**: A string providing the overall conclusion explaining why the preconditions are or are not satisfied.
- **failed_conditions**: A list of predicate strings that are NOT satisfied (empty if all conditions are met).

Table 3: The precondition verification prompt for action checker. It visually validates whether the current scene satisfies all preconditions before executing a planned action, enabling closed-loop error detection.

B Experiment Details

B.1 Task Progress Score

In both the compositional vs. single-VLM experiments and the ablation study, we measure performance using a task progress score derived from a structured action-sequence rubric. The score measures how much of the intended plan was completed, while penalizing unnecessary actions and final goal-check failures, making it more informative than binary success. This structured scoring system allows for consistent evaluation and comparison of task performance.

Concretely, let N be the total number of executed actions, N_{done} the number of actions that match the required action sequence, and N_{extra} the number of redundant actions. Denote $p \in \{0, 1\}$ as a binary penalty for goal-checker error, where $p = 1$ indicates that the goal checker failed to correctly assess

the final task state. We define:

$$\text{Score} = \frac{N_{\text{done}} - N_{\text{extra}}}{N} - 0.1\,p\,.$$

The redundant-action penalty is important because VLMs sometimes introduce redundant goals when generating the problem PDDL, which can trigger unnecessary steps, this behavior is especially common in the crossword task.

B.2 Module Evaluation

We report runtime as an averaged estimate from randomized evaluations conducted across multiple time periods. While latency for cloud-hosted models is largely determined by the provider's service, our goal is to provide an intuitive sense of query-time differences across models, enabling practitioners to make informed deployment choices for robotic agents based on their efficiency requirements.

Table 4: Performance of single-VLM pipelines on the sorting task. Each pipeline uses one VLM for all modules and is evaluated over 20 trials across 6 object setups.

Model Pipeline	Success Rate	Time	Tokens
Gemini 3 Flash	**80.0 %**	334 s	67.0k
Gemini 3 Pro	10.0 %	1283 s	139.0k
Gemini 3 Pro + LangSAM	70.0 %	524 s	80.9k
Gemini Robotics	50.0 %	382 s	**40.6k**
Qwen-VL-Max	0.0 %	–	–
Qwen3-VL-Plus	25.0 %	538 s	43.0k
GPT-5.2 + LangSAM	0.0 %	–	–
GPT-4o + LangSAM	5.0 %	**299 s**	49.6k

B.3 Module Evaluation Details

Data Annotation: We report the data annotation details used in our module-level evaluation.

1. Action Checker. We collect action-checker evaluation data from real rollouts of the PLanAR pipeline, resulting in 61 verification instances across 7 different scenes and 5 tasks. For each model, we run three trials and report the average success rate, latency, and token usage.

2. Grasp Planner. We curate 50 grasp-planning instances covering diverse objects. Executability is labeled from real-world outcomes, where a grasp is considered valid if it picks the correct object and executes without collisions. Each model is evaluated over three runs. Notably, local models frequently collapse to near-constant true/false outputs, indicating limited capability as grasp planners under this protocol.

3. Object Detector. We collect 21 real-world camera inputs and annotate object regions using SAM masks [54] and corresponding bounding boxes. We compute (i) point-in-mask success rate, based on whether the VLM-predicted point lies inside the SAM mask, and (ii) bounding-box IoU, defined as the overlap area divided by the union area between the predicted box and the annotated box. Each model is tested three times and we report averaged results.

4. Goal Checker. For task-level parsing evaluation, we record 16 final-scene cases and ask the VLM to judge whether the task goal is satisfied. Each model is evaluated with three runs. Ground-truth labels are provided by five human annotators, who determine success based on the goal specification and the task prompt.

Module Evaluation Results: We summarize per-module performance with a radar chart in Fig. 22. Each axis corresponds to a module-level metric, and we normalize all scores to $[0, 1]$ so that larger values indicate better performance. We also report efficiency across VLM backends using the average latency in Fig. 23 and token usage in Fig. 24, both aggregated over the same module evaluation runs.

Task	Language Description
Sort	Sort the fruits into the white bin and all other objects into the blue bin.
Stack (3 cubes)	Stack the cubes on the pink plate from bottom to top: Green, Orange, and Blue.
Stack (4 cubes)	Stack the cubes on the pink plate from bottom to top: Green, Yellow, Orange, and Blue.
Crossword	Fill the numbered slots using the provided blocks to solve the crossword puzzle. You do not need to use all blocks or all slots.
Kitchen A	Put the spice bottle into the top drawer and close it.
Kitchen B	Place the blue snack pack in the top drawer, then move the spice bottle from the drawer to the table, and finally close the drawer.
Kitchen C	A chicken leg is in the pot. Take out the chicken leg, place it in the bowl, then put the spice bottle back into the top drawer and close the drawer.

Table 5: Task list and language descriptions.

Human Evaluation Interface: Evaluating the task parser in open-world scenes benefits from a human study, since fully automatic evaluation can rely on brittle, hand-crafted heuristics. We therefore conduct a human study and build a lightweight labeling UI that supports efficient, reproducible evaluation while ensuring fair, blinded comparison across models.

As shown in Fig. 25, participants are shown only (i) the task goal, (ii) the initial scene image at the start of the episode, and (iii) the action sequence produced by the task parser. The model identity is hidden. Participants judge whether the provided action sequence can reasonably achieve the goal under the given initial scene conditions.

We evaluate four task categories: sorting, stacking, crossword, and kitchen. We recruited five participants and collected 468 action-sequence judgments in total. To improve reliability, we perform three full passes over the complete set and report results aggregated across rounds.

Table 5 lists the natural language instructions used in our task parser evaluation, covering the four evaluation categories described above.

B.4 Ablation Study: Action Checker and Grasp Planner

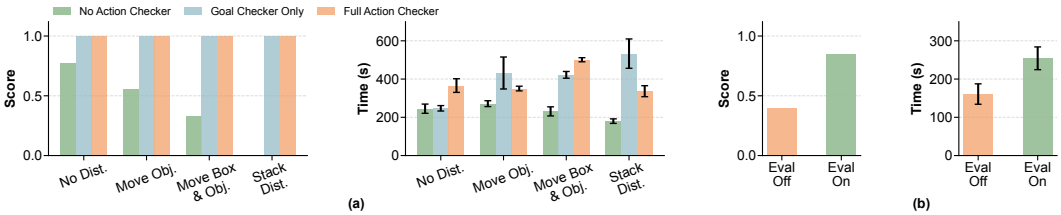


Figure 26: Ablation results for action checking and grasp planning. We report task progress score and execution time under different verification and grasp-planning settings. Error bars denote standard deviation.

In this experiment, we characterize the robustness–efficiency trade-off via targeted ablations of two components. First, we compare three action-checking configurations: no checker, goal checker only, and full action checker, under controlled disturbances (Fig. 26(a)). Second, we ablate the grasp planner by enabling or disabling grasp evaluation, as shown in Fig. 26(b).

We run the action checker ablations on two tasks: sorting, where actions are weakly coupled, and sequential stacking, where actions are strongly dependent and early failures affect all subsequent actions. For the grasp planner, we ablate grasp evaluation on sorting in clustered scenes, where imperfect bounding boxes often include multiple objects and collision-prone grasps are common.

Two patterns emerge. First, grasp evaluation is critical in cluttered scenes. When it is disabled, task success drops sharply because the system is more likely to execute collision-prone grasps or pick the wrong object when the detected workspace contains multiple nearby items.

Second, verification frequency determines when errors are detected. Although a full action checker increases execution time, it can correct mistakes by detecting failures early. In stacking, disabling verification leads to zero task success because early errors propagate and the final stack violates the target order. Using only a goal checker delays error detection until task completion: the system continues stacking despite previous mistakes, producing an invalid intermediate stack and requiring replanning from a more complex state, which results in unnecessary actions and increased execution time. Full action checker instead enables early detection and timely recovery.

Overall, these results point to a practical guideline: sparse verification improves evaluation efficiency, whereas dense verification is required when tasks involve strong dependencies or active disturbances.

B.5 Ablation Experiment Protocol

For the action-checker ablation, we evaluate performance using the task progress score in Appendix B.1 and measure efficiency by runtime under controlled human disturbances. We compare three action-checker configurations: no action checker, a goal checker only, and the full action checker. The goal-checker-only configuration performs a single verification at the end of execution, such as checking whether the final goal is satisfied after all planned actions complete. In contrast, the full action checker verifies outcomes step by step after each action.

We use the following disturbance protocols.

1. No disturbance, sorting. We execute the sorting task without human intervention, placing food items into the box and toys into the bowl.

2. One disturbance, sorting. During the first pick, we move the target object to induce a misgrasp and simulate grasp failure.

3. Two disturbances, sorting. In addition to the first-pick disturbance, we move the target placement location during the final place action to simulate misplacement under a shifted goal configuration.

4. One disturbance, stacking. To test a sequential task where later outcomes depend on earlier actions, we move the target block during the first grasp to induce a disturbance.

B.6 VLA Fine-tuning Setup

We fine-tune the pretrained $\pi_{0.5}$ VLA policy [4] on two tabletop manipulation tasks: sorting and stacking. Training configurations are provided in Table 6.

For sorting, we collect 40 human demonstrations, evenly split into two instruction variants: 20 episodes of "Grab the food to the bowl" and 20 episodes of "Grab the toys to the bin". For stacking, we collected 30 demonstrations consisting of 10 episodes for each of three stacking orders: green-yellow-orange, yellow-green-orange, and orange-yellow-green.

We initialize the policy from the $\pi_{0.5}$-base checkpoint. To retain pretrained visual representations, we freeze the vision encoder and fine-tune only the reasoning and action components via Low-Rank Adaptation (LoRA) [55]. Specifically, we apply LoRA to the linear projection layers in the Attention and FFN modules for both the VLM backbone and the flow-matching action expert. For all methods, we use the same perception setup consisting of one global camera and one wrist camera to ensure a fair comparison.

Table 6: VLA fine-tuning configuration. We apply LoRA to attention and FFN blocks for both the VLM backbone and the flow-matching action expert.

Component	**Setting**
Initialization	$\pi_{0.5}$-Base
VLM Backbone	Gemma-2B
Vision Encoder	Frozen
LoRA Target	Attention & FFN Linear Projections
LoRA (r, α)	$(16, 16)$
Action Expert	Gemma-300M
LoRA Target	Attention & FFN Linear Projections
LoRA (r, α)	$(32, 32)$
Training Steps	10,000
Batch Size	64
Hardware	$1\times$ NVIDIA A6000 (48 GB)
Training Duration	$\sim$38 Hours

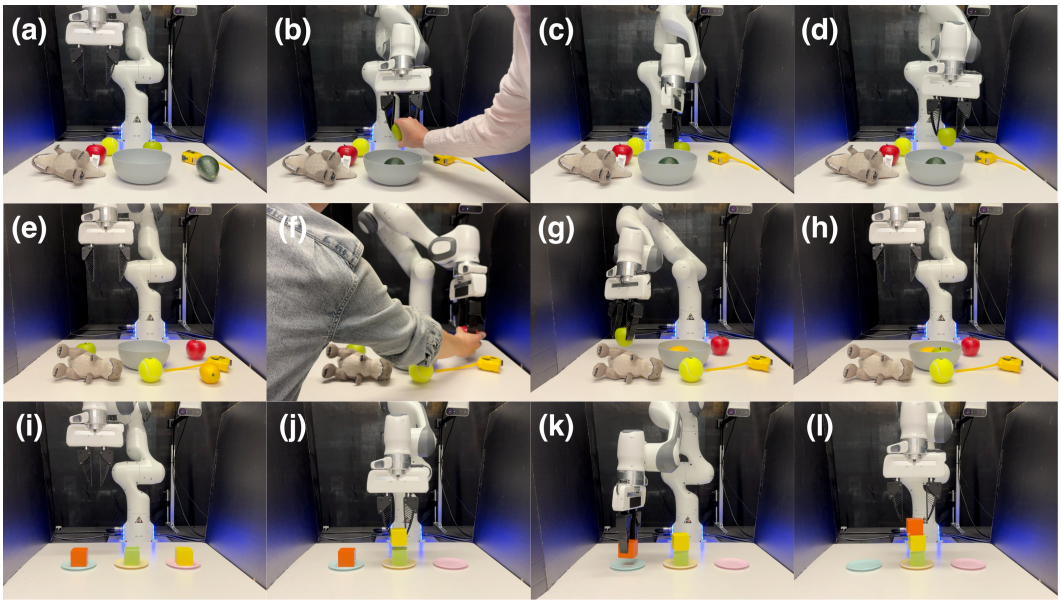


Figure 27: Comparison between PLanAR and TiPToP on the Franka Panda. (a–d) PLanAR on disturbed fruit sorting. After human disturbance during the second grasp, PLanAR detects the failed effect, retries the grasp, detects a second failed grasp caused by an inaccurate grasp pose, and replans again to complete the task. (e–h) TiPToP on the same disturbed fruit-sorting setup. Although TiPToP generates a sorting plan through cuTAMP, it does not detect the disturbed second grasp and terminates with only two fruits sorted. (i–l) PLanAR transfers to block stacking on the Franka Panda, while TiPToP does not generate an executable stacking plan in our setup.

B.7 Comparison of PLanAR and TiPToP

Fig. 27 provides a comparison between PLanAR and TiPToP on the Franka Panda. In the fruit-sorting task, PLanAR successfully transfers the same high-level reasoning, verification, and replanning logic to the Franka platform. As shown in Fig. 27(a–d), we introduce a human disturbance during the second grasp by removing the grasped fruit and returning it to the table. PLanAR detects that the expected grasp effect is not satisfied and retries the grasp. Although the first retry also fails due to

an inaccurate grasp pose, PLanAR detects this failure again, replans a second retry, and eventually completes the task by sorting all three fruits.

In contrast, TiPToP can generate a fruit-sorting plan with cuTAMP, as shown in Fig. 27(e–h), but it does not detect the disturbed second grasp. As a result, it continues execution and terminates with only two fruits sorted. We further evaluate block stacking in Fig. 27(i–l). PLanAR completes the stacking task on the Franka Panda using the same planning-language-grounded interface, whereas TiPToP does not generate an executable stacking plan with its motion planner in our setup. These examples illustrate that PLanAR's explicit verification and replanning are useful under disturbances and failed grasps, and that its primitive-based execution interface can provide a practical transfer path across robot embodiments.

B.8 In-the-Wild Deployment on UR5e

To evaluate PLanAR beyond controlled tabletop settings, we deploy the UR5e platform across diverse real-world scenes, including laboratory, indoor, and outdoor environments. As shown in Fig. 28, these deployments qualitatively demonstrate that the platform can execute long-horizon manipulation tasks under varied backgrounds, lighting conditions, clutter, and object arrangements. They also provide real-world rollout data used for module-level evaluation, including visual grounding, action checking, grasp planning, and goal checking.

Figure 28: Qualitative in-the-wild deployment of PLanAR on the UR5e platform across diverse scenes.

PDDL Problem: stacking

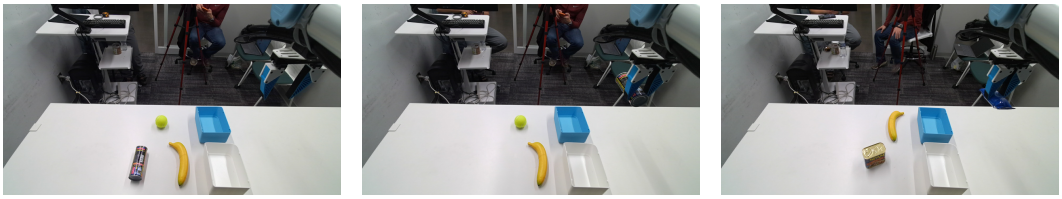
```
(define (problem stacking-stack-cubes-on-pink-
    plate)
  (:domain vlmrobobench)

  (:objects
    green-cube-1 - green-cube
    yellow-cube-1 - yellow-cube
    orange-cube-1 - orange-cube
    blue-cube-1 - blue-cube
    pink-plate-1 - pink-plate
    turquoise-plate-1 - turquoise-plate
    cream-plate-1 - cream-plate
  )

  (:init
    (hand-empty)
    (on-top-of yellow-cube-1 blue-cube-1)
    (on-top-of blue-cube-1 cream-plate-1)
    (on-top-of green-cube-1 orange-cube-1)
    (on-top-of orange-cube-1 turquoise-plate-1)
    (clear yellow-cube-1)
    (clear green-cube-1)
    (clear pink-plate-1)
  )

  (:goal
    (and
      (on-top-of green-cube-1 pink-plate-1)
      (on-top-of yellow-cube-1 green-cube-1)
      (on-top-of orange-cube-1 yellow-cube-1)
      (on-top-of blue-cube-1 orange-cube-1)
    )
  )
)
```

Figure 8: The PDDL Problem File Generated by Gemini 3 Flash. Stack the cubes on the pink plate from bottom to top: Orange Blue Yellow and Green cubes

Action Seqence Solved by PDDL Planner

```
(pick green-cube-1 orange-cube-1);
(place-on-object green-cube-1 yellow-cube-1);
(pick orange-cube-1 blue-plate-1);
(place-on-object orange-cube-1 pink-plate-1);
(pick green-cube-1 yellow-cube-1);
(place-on-object green-cube-1 blue-plate-1);
(pick yellow-cube-1 blue-cube-1);
(place-on-object yellow-cube-1 green-cube-1);
(pick blue-cube-1 white-plate-1);
(place-on-object blue-cube-1 orange-cube-1);
(pick yellow-cube-1 green-cube-1);
(place-on-object yellow-cube-1 blue-cube-1);
(pick green-cube-1 blue-plate-1);
(place-on-object green-cube-1 yellow-cube-1)
```

Figure 9: Action sequence obtained by solving the updated domain file with the generated problem file.



(a) Precondition check (pick can-1) (b) Effect check (pick can-1) (c) Effect check (pick oreo-pack-1)

Figure 10: Precondition/effect checks for pick actions across objects.

Gpt-5.2 Precondition Check Result: pick can-1


```
{
    "success": true,
    "reasoning": "Both preconditions     (hand-empty) and (on-table can-1)     are
        satisfied. The robot gripper is empty and ready to pick, and the can is resting
         on the table as required. Since the action uses an AND operator, all
        conditions must be met, and they are.",
    "elapsed_time": 11.282440900802612,
    "detailed_analysis": [
        {
            "condition": "(hand-empty)",
            "satisfied": true,
            "observation": "The robot gripper is visible in the upper right corner and
                is not holding any object; it is open and positioned above the table,
                indicating it is empty."
        },
        {
            "condition": "(on-table can-1)",
            "satisfied": true,
            "observation": "The cylindrical can (can-1) is clearly placed directly on
                the white table surface, with no other objects on top of it or between
                it and the table."
        }
    ]
}
```


Figure 11: Precondition check for picking can (success, GPT-5.2).

Gpt-5.2 Effect Check Result: pick can-1


```
{
    "success": false,
    "reasoning": "All three expected effects must be achieved for success. However,
        none of them are satisfied: the robot is not holding any object, the hand is
        empty (contradicting 'not (hand-empty)'), and no object has been lifted off the
         table. Thus, the action '(pick oreo-pack-1)' was not successfully executed.",
    "elapsed_time": 14.711339473724365,
    "failed_conditions": [
        "(holding ?x)",
        "(not (hand-empty))",
        "(not (on-table ?x))"
    ],
    "detailed_analysis": [
        {
            "effect": "(holding ?x)",
            "achieved": false,
            "observation": "The robot gripper is not holding any object. The gripper is
                 open and empty, with no visible item grasped."
        },
        {
            "effect": "(not (hand-empty))",
            "achieved": false,
            "observation": "The gripper is empty, so the condition '(not (hand-empty))'
                 is false. The hand is indeed empty."
        },
        {
            "effect": "(not (on-table ?x))",
            "achieved": false,
            "observation": "Since no object is being held, there is no '?x' that has
                been removed from the table. Therefore, this condition cannot be
                satisfied as no object has been picked up."
        }
    ]
}
```


Figure 12: Effect check for picking can (fail, GPT-5.2).

Qwen-VL-Plus Effect Check Result: pick can-1


```
{
    "success": false,
    "reasoning": "The action 'pick can-1' was executed, but there is no visible can-1
        in the scene. The table contains a banana, a tennis ball, and two empty
        containers (blue and white), but no can. Therefore, none of the expected
        effects -- (holding ?x), (not (hand-empty)), or (not (on-table ?x)) -- can be
        verified as achieved because the target object 'can-1' does not appear to exist
         or be present in the environment.",
    "elapsed_time": 16.5667142868042,
    "failed_conditions": [
        "(holding ?x)",
        "(not (hand-empty))",
        "(not (on-table ?x))"
    ]
}
```


Figure 13: Effect check for picking can (fail, Qwen-VL-Plus).

Gpt-5.2 Effect Check Result: pick oreo-pack-1


```
{
    "success": false,
    "reasoning": "All three expected effects must be achieved for success. However,
        none of them are satisfied: the robot is not holding any object, the hand is
        empty (contradicting 'not (hand-empty)'), and no object has been lifted off the
         table. Thus, the action '(pick oreo-pack-1)' was not successfully executed.",
    "elapsed_time": 14.711339473724365,
    "failed_conditions": [
        "(holding ?x)",
        "(not (hand-empty))",
        "(not (on-table ?x))"
    ],
    "detailed_analysis": [
        {
            "effect": "(holding ?x)",
            "achieved": false,
            "observation": "The robot gripper is not holding any object. The gripper is
                open and empty, with no visible item grasped."
        },
        {
            "effect": "(not (hand-empty))",
            "achieved": false,
            "observation": "The gripper is empty, so the condition '(not (hand-empty))'
                is false. The hand is indeed empty."
        },
        {
            "effect": "(not (on-table ?x))",
            "achieved": false,
            "observation": "Since no object is being held, there is no '?x' that has
               been removed from the table. Therefore, this condition cannot be
               satisfied as no object has been picked up."
        }
    ]
}
```


Figure 14: Effect check for picking oreo pack (fail, GPT-5.2).

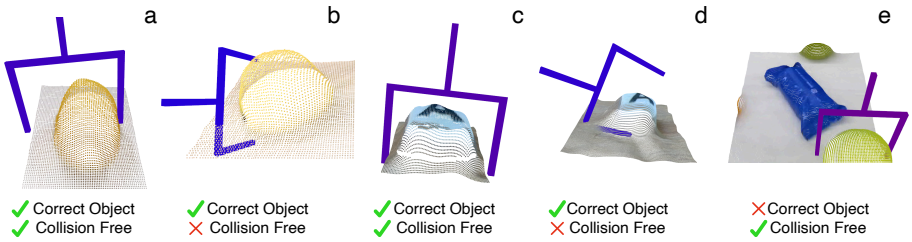


Figure 15: Evaluation of proposed grasp poses. Ground-truth labels are provided below each example.

Gemini 3 Flash Grasp Planning Result: pick lemon-1


```
"evaluation": {
    "action": "EXECUTE_GRASP",
    "object_correctness": {
        "is_correct_object": true,
        "confidence": 0.98,
        "reasoning": "The gripper frame is accurately centered over the yellow
            point cloud representing the lemon. The fingers are wide enough to
            enclose the object's bulk."
    },
    "collision_check": {
        "overlaps_with_surface": false,
        "confidence": 0.92,
        "reasoning": "The gripper fingers terminate above the table surface.
            While the right finger is close to the table plane, there is no
            evidence of deep penetration that would cause a failure."
    },
    "overall_reasoning": "The grasp is correctly targeted at the lemon and
        maintains a safe distance from the table surface. It is a functional
        grasp pose for the specified task.",
    "confidence": 0.95,
    "rejection_reason": null,
    "elapsed_time": 5.428056955337524
}
```


Figure 16: Gemini 3 Flash output for example a in Fig. 15, planning the pick lemon-1 action.

Gemini 3 Flash Grasp Planning Result: pick lemon-1


```
"evaluation": {
    "action": "REJECT_GRASP",
    "object_correctness": {
        "is_correct_object": true,
        "confidence": 0.95,
        "reasoning": "The gripper is correctly positioned around the yellow point
            cloud cluster, which represents the 'lemon_collision' target object."
    },
    "collision_check": {
        "overlaps_with_surface": true,
        "confidence": 0.98,
        "reasoning": "The bottom finger of the gripper and a significant portion of
             the gripper bridge (the vertical bar) are clearly submerged beneath
            the table surface points (the tan point cloud). Visible surface points
            are rendered on top of the gripper's blue geometry, indicating that
            more than half of the lower finger is deeply intersecting with the
            table plane."
    },
    "overall_reasoning": "Although the gripper is correctly targeting the lemon,
        the grasp pose is physically impossible as the lower finger and part of the
         gripper's base are positioned below the level of the table surface. This
        deep penetration into the solid table surface would likely cause a hardware
         failure or damage the robot.",
    "confidence": 0.98,
    "rejection_reason": "Severe collision with the table surface; the lower gripper
         finger is almost entirely embedded in the table plane.",
    "elapsed_time": 20.040879011154175
}
```


Figure 17: Gemini 3 Flash output for example b in Fig. 15, planning the pick lemon-1 action.

Gemini 3 Flash Grasp Planning Result: pick block-a-1


```
"evaluation": {
        "action": "EXECUTE_GRASP",
        "object_correctness": {
            "is_correct_object": true,
            "confidence": 0.98,
            "reasoning": "The gripper frames the central mass of the point cloud, which
                 represents the target block. It is well-centered and encompasses the
                bulk of the object."
        },
        "collision_check": {
            "overlaps_with_surface": false,
            "confidence": 0.9,
            "reasoning": "The gripper fingers are positioned on the sides of the object
                . While the tips reach towards the bottom of the point cloud, there is
                no evidence of deep penetration into the table surface that would
                constitute a catastrophic collision."
        },
        "overall_reasoning": "The grasp pose is correctly targeted at the specified
            object and appears to be collision-free relative to the visible point cloud
             data. The alignment is suitable for a successful pick operation.",
        "confidence": 0.95,
        "rejection_reason": null,
        "elapsed_time": 5.686762571334839
    }
```


Figure 18: Gemini 3 Flash output for example c in Fig. 15, planning the pick block-a-1 action.

Gpt-5.2 Grasp Planning Result: pick block-a-1

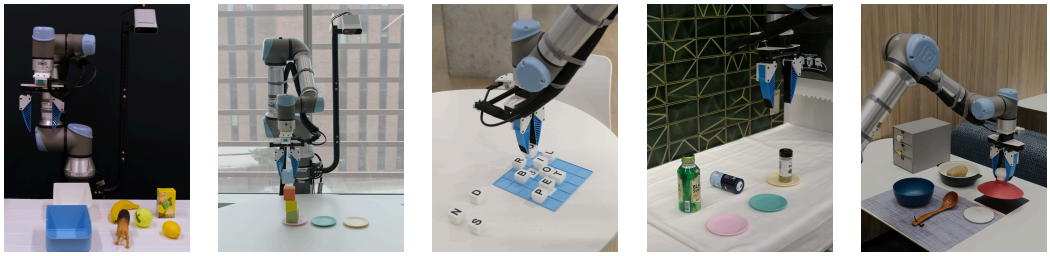

```
"evaluation": {
        "action": "REJECT_GRASP",
        "object_correctness": {
            "is_correct_object": false,
            "confidence": 0.74,
            "reasoning": "The grasp frame (blue/purple pose indicator) is positioned
                primarily over the table/ground point cloud and appears offset to the
                side/below the main bulk of the target point cluster. The densest,
                raised object mass (block_a) is not centered within the implied gripper
                 workspace; instead, the pose sits near a low-lying region, suggesting
                it would close on background/table points rather than the object."
        },
        "collision_check": {
            "overlaps_with_surface": false,
            "confidence": 0.62,
            "reasoning": "While the pose is close to the table surface, the
                visualization does not show a clear deep penetration of more than half
                a finger length into a solid plane; it looks like it would at most
                graze the fuzzy surface points. The primary failure is mis-targeting
                rather than a catastrophic collision."
        },
        "overall_reasoning": "The grasp pose is not aligned with or centered on the
            main elevated cluster corresponding to block_a, so it would likely miss the
             object and/or clamp on table/background points. No clearly destructive
            table intersection is evident, but object correctness is insufficient to
            execute.",
        "confidence": 0.71,
        "rejection_reason": "Grasp pose is not targeting the main body of block_a; it
            is offset toward the table/background rather than framing the object.",
        "elapsed_time": 7.609699964523315
    }
```


Figure 19: GPT-5.2 output for example d in Fig. 15, planning the pick block-a-1 action.

Gemini 3 Pro Grasp Planning Result: pick blue-snack-pack-1


```
"evaluation": {
        "action": "REJECT_GRASP",
        "object_correctness": {
            "is_correct_object": false,
            "confidence": 1.0,
            "reasoning": "The grasp pose is positioned significantly offset to the
                right of the target 'blue_snack_pack'. The gripper fingers frame the
                white table surface and empty space next to the object, failing to
                enclose the bulk or center of the blue snack pack."
        },
        "collision_check": {
            "overlaps_with_surface": false,
            "confidence": 0.9,
            "reasoning": "While the gripper is positioned low near the table plane,
                there is no clear evidence of deep, destructive intersection with a
                solid obstacle. The primary failure is spatial targeting."
        },
        "overall_reasoning": "The grasp is completely invalid because it misses the
            target object. The gripper's closing region is located over the table
            surface to the side of the snack pack, rather than around the object itself
            .",
        "confidence": 1.0,
        "rejection_reason": "Gripper is completely missing the target object",
        "token_usage": 3771,
        "elapsed_time": 22.42589521408081
    }
```


Figure 20: Gemini 3 Pro output for example e in Fig. 15, planning the pick blue-snack-pack-1 action.



Sorting Stacking Crossword Reorientation Kitchen

Figure 21: Manipulation tasks used for real-world evaluation. These tasks are selected to cover a range of challenges, including spatial reasoning, vision-language grounding, and long-horizon planning.

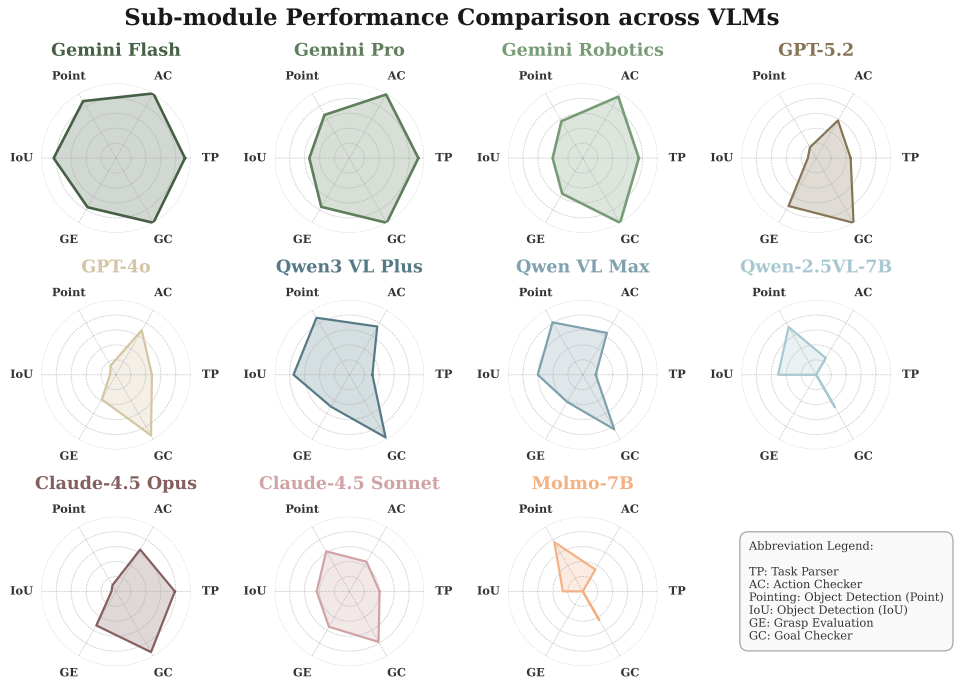


Figure 22: Modular evaluation radar chart. Normalized per-module performance across the pipeline (higher is better). Each axis reports the corresponding module score.

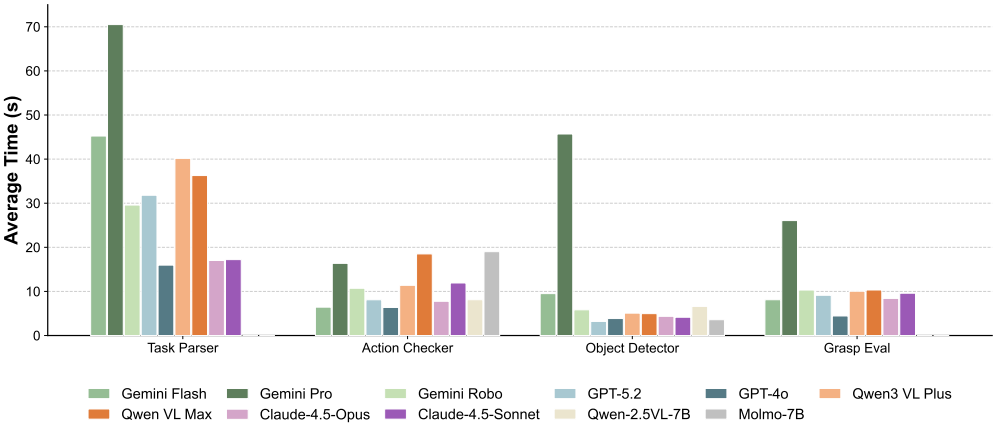


Figure 23: Latency comparison across VLMs. Average runtime per module evaluation query.

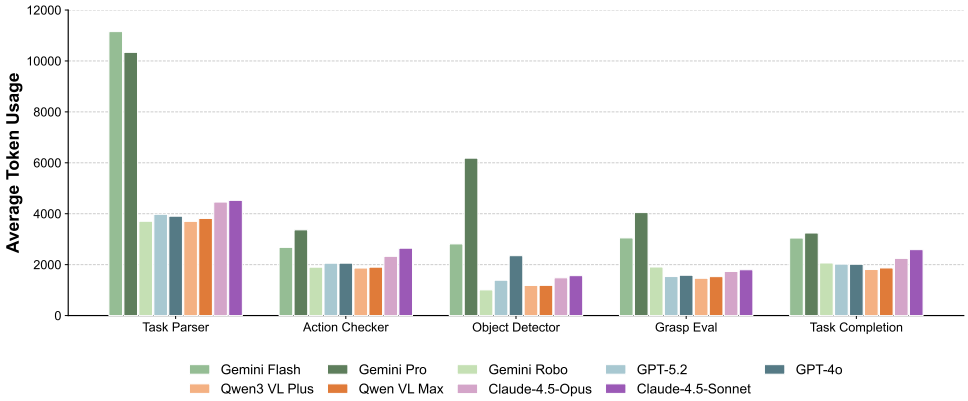


Figure 24: Token usage comparison across VLMs. Average token consumption per module evaluation query.

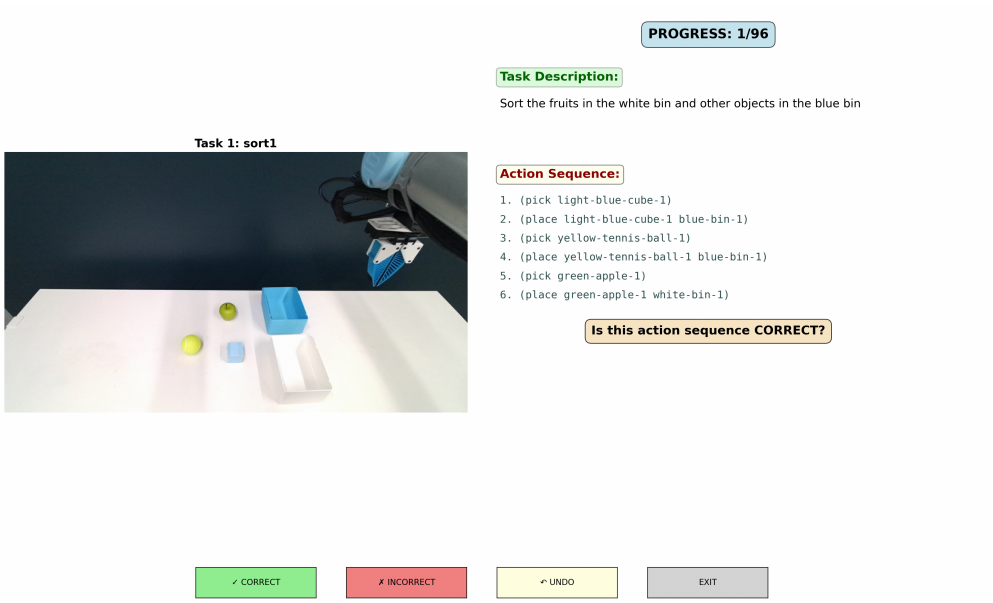


Figure 25: Human evaluation UI.