SOLVING ROBOTICS PROBLEMS IN ZERO-SHOT WITH VISION-LANGUAGE MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Wonderful Team, a multi-agent Vision Large Language Model (VLLM) framework designed to solve robotics problems in a zero-shot regime. In our context, zero-shot means that for a novel environment, we provide a VLLM with an image of the robot's surroundings and a task description, and the VLLM outputs the sequence of actions necessary for the robot to complete the task. Unlike prior work that requires fine-tuning parts of the pipeline – such as adjusting an LLM on robot-specific data or training separate vision encoders – our approach demonstrates that with careful engineering, a single off-the-shelf VLLM can autonomously handle all aspects of a robotics task, from high-level planning to low-level location extraction and action execution. Crucially, compared to using GPT-4o alone, Wonderful Team is self-corrective and capable of iteratively fixing its own mistakes, enabling it to solve challenging long-horizon tasks. We validate our framework through extensive experiments, both in simulated environments using VIMABench and in real-world settings. Our system showcases the ability to handle diverse tasks such as manipulation, goal-reaching, and visual reasoning—all in a zero-shot manner. These results underscore a key point: vision-language models have progressed rapidly in the past year and should be strongly considered as a backbone for many robotics problems moving forward.

1 INTRODUCTION

Advancements in Large Language Models (LLMs) and Vision-Language Models (VLLMs) have brought us closer to enabling robots to perform complex tasks based solely on natural language instructions, without prior training. By integrating vision and language, VLLMs allow robots to intuitively understand their environments, leveraging real-world priors from large-scale data. However, developing a general-purpose robotic system capable of executing complex tasks in dynamic settings remains challenging. Such systems need to perceive surroundings, utilize appropriate skills, and achieve long-horizon subgoals. This raises a crucial question: **Can these models be adapted to solve robotic tasks in unstructured environments without any training?**

Current approaches in language-conditioned robotics often separate the problem into high-level planning and low-level perception-action execution, utilizing distinct modules for each component. While this separation can facilitate zero-shot operation, it may hinder seamless integration between perception and action, especially when modules are disconnected.

High-Level Planning with Predefined Task Modules: Many methods focus on high-level planning using LLMs or VLLMs, decomposing tasks into subtasks but relying on predefined task modules or APIs for action execution, which are not directly executable without prior knowledge or training (Hu et al., 2023; Huang et al., 2022b; Liang et al., 2023).

Low-Level Coordinate Generation with Separate Vision Models: Other approaches generate low-level coordinates using separate vision models for perception, often relying on predefined or fine-tuned vision APIs. While leveraging off-the-shelf models like Convolutional Neural Networks (CNNs)(Ichter et al., 2022; Mees et al., 2023), CLIP(Bucker et al., 2023; Huang et al., 2022c), Vision Transformer (ViT) variants (Huang et al., 2023b; Stone et al., 2023; Jiang et al., 2023), or LangSAM (Kwon et al., 2024) has shown promise in zero-shot capabilities, these methods still face limitations. The reliance on separate perception systems can fail to fully capture the environmental context required for precise planning and action generation.

These limitations hinder the seamless integration of perception and action, as vision models like CLIP, which primarily offer class-level predictions, lack the deep environmental understanding needed for complex, context-specific tasks. Similarly, while LangSAM can segment objects based on language prompts, it struggles with precise object identification in complex scenes or when handling abstract instructions that require deeper comprehension. As a result, these models perform well with easily identifiable objects but face challenges when handling abstract or environment-specific tasks, which significantly limits their ability to help LLMs accurately ground environmental context and generate actionable outputs. The separation of planning and perception hinders the seamless integration of perception and action in decision-making. However, with the multimodal capabilities of modern VLLMs, this division may no longer be necessary. **In this paper, we introduce Wonderful Team: a zero-shot, single-model, multi-agent system that unifies planning and perception within a VLLM framework using interconnected specialized agents.** This integrated approach enables end-to-end reasoning and execution without relying on external modules or fine-tuning, effectively addressing the limitations of previous modular methods.

Our key contributions include:

- **Zero-Shot Coordinate-Level Control in Complex Robotics Tasks:** Our system operates without any prior training, fine-tuning, or environment-specific prompts, successfully handling diverse tasks in both simulated and real-world environments. It delivers precise, coordinate-level control for robotic execution, outperforming methods that rely on coarse object-level or sub-task-level instructions.
- **Introducing a Multi-Agent VLLM Framework to Overcome Previous Limitations:** We have developed a novel multi-agent structure within a single VLLM, where specialized agents collaboratively handle various aspects of robotic tasks, from high-level planning to low-level execution. By integrating perception and action, and employing a divide-and-conquer approach with reflection capabilities, we address the shortcomings of previous models, including issues with context-aware object identification, precise localization, and handling multiple instances of the same object.
- **Empirical Validation through Extensive Experiments and Ablation Studies:** We validate our framework with comprehensive experiments in both simulation (VIMABench) and real-world settings. Our results show significant performance improvements over existing methods, including those that require training. We also conduct thorough ablation studies to examine the effects of different agents and configurations, highlighting the critical role of the multi-agent system in achieving optimal performance.

Demonstration videos of the robotic policies in action, along with the code, can be accessed on our project website.

2 Motivating Examples

Developing robotic systems that can understand and execute complex tasks in unstructured environments remains a significant challenge. Existing frameworks often employ a Large Language Model (LLM) as a text planner combined with a separate vision model (e.g., CLIP, OWL-ViT, LangSAM) to perceive the environment. While this modular approach seems logical, it faces critical limitations when applied to intricate, context-dependent tasks.

2.1 Can an LLM as a Planner with a Separate Vision Model Find Objects?

Not Always. There are limitations at both the planning and perception levels:

At the *planning level*, non-vision LLMs cannot generate meaningful plans for ambiguous prompts that rely on environmental context. For example, consider the task: *“Rank the fruits from most expensive to cheapest.”* Without visual input to identify the fruits and their prices, the LLM cannot accurately rank them, nor generate useful queries for the vision model.

At the *perception level*, vision models also have limitations in context-aware perception. A notable prior work is the *Trajectory Generator* (Kwon et al., 2024), which uses GPT as a text planner and LangSAM as the vision model. In this approach, GPT extracts the objects to segment from the task prompt and passes them to LangSAM for object identification and segmentation. As illustrated in Figure 1, LangSAM fails to correctly identify or segment all intended objects based on the prompt. While this example highlights several challenges inherent in using separate vision models for complex tasks, it does not capture the full scope of limitations, which are discussed in detail below:

1. Difficulty with Less Common and Non-Segmented Objects: LangSAM struggles to identify uncommon objects (e.g., robot grippers, box lids) and abstract regions that cannot be clearly segmented. When objects are less prominent in the scene or when boundaries are not well-defined, LangSAM fails to provide accurate identification or spatial understanding.

2. Misinterpretation of Spatial and Positional Instructions: LangSAM often misinterprets vague spatial instructions like "pick up the rightmost object" due to its lack of precise spatial reasoning. In multi-instance scenarios, positional references like "the middle can" are challenging because the model frequently miscounts objects, leading to incorrect identification.

3. Lack of Contextual Awareness and Differentiation: LangSAM lacks the contextual understanding necessary to distinguish between relevant objects for manipulation and other elements in the scene. For instance, it may mistakenly select parts of the robot arm itself, failing to identify the intended target due to a lack of contextual awareness.

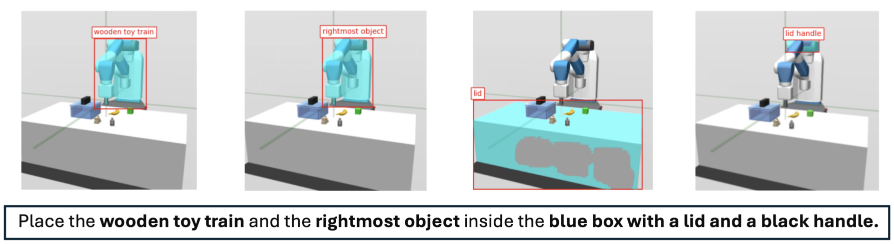


Figure 1: Examples of LangSAM's detection failures in simulated environments. The **bolded text** within the prompts represents the objects extracted by GPT and passed to LangSAM.

Can These Issues Be Fixed?

Not within the current framework. Even with enhanced reasoning and replanning, we are unable to fully address LangSAM's limitations because the LLM lacks the capability to detect, notice, or correct errors originating from the vision model. If the initial perception is flawed, the LLM cannot adjust or rectify these mistakes, resulting in a disconnect between perception and reasoning within this setup.

However, recent advancements in VLLMs present a potential solution, as they are designed to handle both visual reasoning and context understanding. This brings us to the question:

2.2 Could simply replacing LangSAM with a VLLM resolve these issues?

Partially; a VLLM may improve context comprehension, but it fails to match the precision that LangSAM already provides.

To provide a clearer context for spatial reasoning, we first introduce pixel coordinates as a reference framework (see Figure 2). Without this grid overlay, even humans might struggle to describe relative locations accurately in a complex scene.

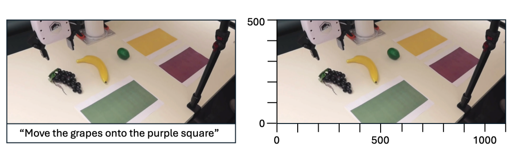


Figure 2: An example scenario with overlaid pixel coordinates.

However, there are still notable challenges with this framework:

1. Imprecise Spatial Understanding: Recent VLLMs can generate more accurate approximate locations, but they still lack the precision required for effective robotic manipulation. In our ablation experiments, 90% of the coordinates were close to the target (Table 6), yet only 33% (GPT-4o) were accurate enough to be directly actionable (Table 5).

2. Difficulty with Complex Instructions: Tasks that require understanding spatial relationships or handling multiple objects can overwhelm the reasoning capabilities. **Observation 1: VLLMs Can Recognize and Diagnose Their Own Errors**

VLLMs have the ability to detect mistakes in their outputs and adjust them upon review. For example, when asked to locate a cluster of grapes, the model may initially provide an imprecise answer, but can correct it when prompted to reassess (see Figure 3). Table 7 shows GPT-4o's 97% success in classifying bounding boxes, highlighting its self-assessment abilities. This suggests VLLMs can iteratively refine outputs, even from initially imprecise coordinates.

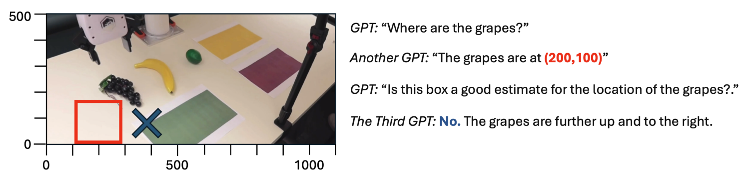


Figure 3: An example of multiple VLLMs working together to recognize and correct an error in object positioning upon review.

Observation 2: VLLMs Can Self-Correct Through Reflection

VLLMs can iteratively refine their outputs based on feedback, a process known as *reflection*. Over several iterations, they improve their estimation of an object's position, moving closer to the correct target (see Figure 4).

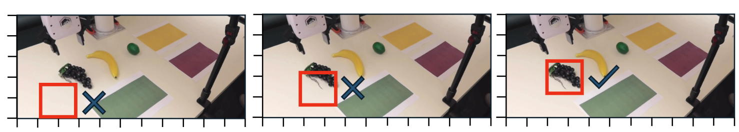


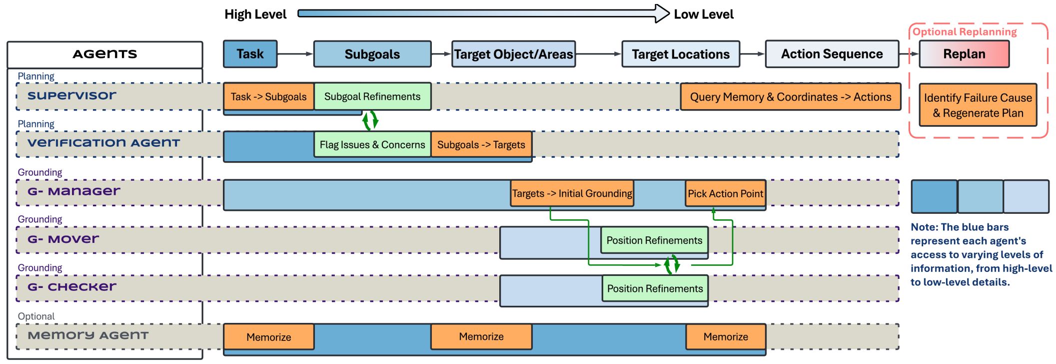
Figure 4: An VLLM improving its estimation of the grapes' position over several iterations.

While using a VLLM alone naively is insufficient, these observations reveal the potential to address its limitations by leveraging its self-correction capabilities in a structured way.

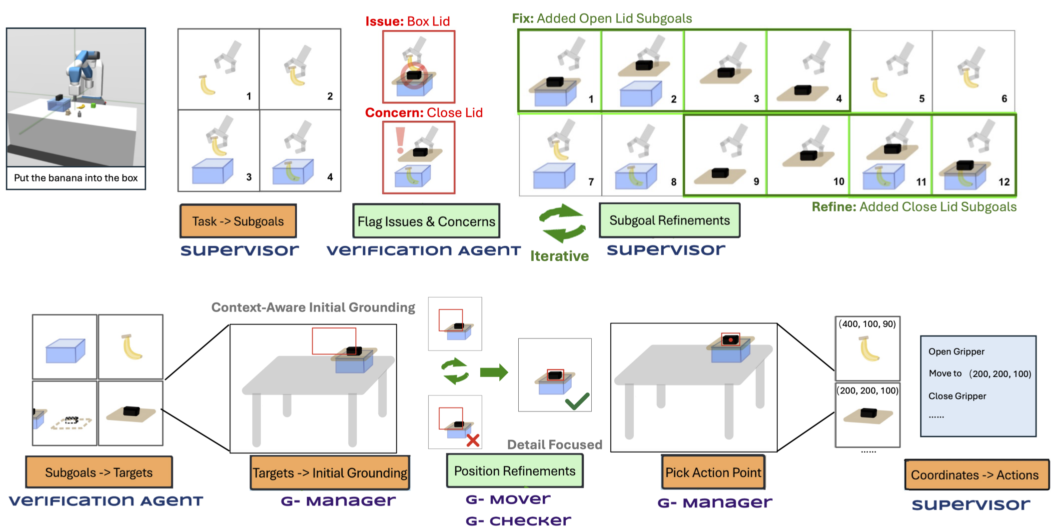
3 WONDERFUL TEAM

Building on these insights, we propose a novel pipeline for robotics that leverages specialized agents, each responsible for a distinct part of the reasoning process within a structured framework. By combining the strengths of Vision-Language Models (VLLMs) and breaking down complex tasks into manageable components, each agent can focus on a specific role, resulting in more precise and reliable robotic control. As illustrated in Figure 5, our multi-agent framework defines the distinct roles of each agent, the flow of information from high-level tasks to low-level actions, and their collaborative efforts in executing tasks effectively.

Each agent in our system is designed to address specific challenges in robotic tasks. For example, in Figure 5(b), when the robot is instructed to "put the banana into the box," the initial plan generated by the Supervisor agent often overlooks obstacles like the box's lid. This is where the Verification agent plays a critical role. Its reflection process involves reviewing the subgoal plan, checking for potential issues such as physical constraints or incomplete steps, and cross-referencing this plan with the current state of the environment. If an issue, like the lid blocking access to the box, is detected, the Verification agent raises this concern to the Supervisor. This early feedback allows the system to refine the plan before executing any action. Unlike the replanning process, which occurs at the end of the pipeline if a task fails, the Verification agent catches errors early to prevent failures and avoid costly adjustments later. This proactive approach enhances the robustness and adaptability of the robotic control.



(a) This figure illustrates the agent roles and information flow within our pipeline, moving from high-level tasks to low-level actions. The blue bars indicate each agent's level of information access. For instance, the Grounding Manager has a broad overview, encompassing both the task and subgoals, while the Mover and Checker agents focus only on specific details within their target areas, without managing the entire task context.



(b) A symbolic example illustrating the framework in (a).

Figure 5: Illustration of our multi-agent framework and a symbolic example showcasing agent roles, information flow, and collaborative task execution.

The Grounding team then takes over to refine the coordinates for each target, ensuring precise and collision-free movements. The Mover and Checker agents collaborate through an iterative process of adjusting positional groundings. Figure 4 provides an example of the Grounding team in action. The separation of tasks into a multi-agent system proves advantageous, as it allows each agent to focus on its distinct responsibilities with varying levels of access to critical information. For a detailed discussion on the benefits of this multi-agent approach, refer to Appendix E.4.

Are all parts of the Wonderful Team necessary? Ablation studies reveal that all components of the Wonderful Team are essential. Removing memory agents leads to failures, such as mistaking irrelevant objects for targets, while omitting grounding members results in inaccurate coordinates. A supervisor-only setup works for simple tasks but fails with complex ones, lacking precision and corrective processes. Appendix C provides detailed analysis, and Table 4 in the appendix shows the impact on success rates when specific agents are removed.

4 RELATED WORK

Recent advancements in robotics and artificial intelligence have integrated Large Language Models (LLMs) and Vision-Language Models (VLMs) into robotic systems. Our work builds upon and differs from several key areas in this evolving landscape.

Foundation Models in Robotics: Foundation models, trained on vast internet-scale datasets, have demonstrated strong zero-shot capabilities across various tasks. LLMs like GPT-3 (Brown et al., 2020), LLaMA (Touvron et al., 2023), and ChatGPT have excelled in generating human-like text, understanding natural language instructions, and performing extensive reasoning and planning. VLMs extend these capabilities by incorporating visual understanding. In robotics, these models offer the potential to endow robots with real-world priors and advanced reasoning abilities without extensive task-specific training.

Language Models Empowering Robotics: Prior work has leveraged natural language to enhance robotic learning and adaptation. Early approaches equipped agents with learned language embeddings, requiring large amounts of training data (Bing et al., 2023; Jiang et al., 2023). Others focused on connecting language instructions with low-level action primitives to solve long-horizon tasks (Hu et al., 2023; Huang et al., 2022b; Liang et al., 2023). While effective in specific contexts, these methods often struggle to generalize to new tasks without retraining. Foundation models like RT-1 (Brohan et al., 2022) and RT-2 (Brohan et al., 2023) have advanced versatile robotic systems, but they still require significant training to achieve robust performance across diverse tasks.

Zero-Shot and Few-Shot Approaches: Recent studies have explored zero-shot and few-shot solutions for robotic planning and manipulation tasks (Huang et al., 2022a; Liang et al., 2023; Huang et al., 2022b;c; Zeng et al., 2023; Singh et al., 2023; Vemprala et al., 2023; Gu et al., 2023). These approaches aim to handle unseen scenarios without prior training, primarily focusing on high-level planning. However, they often rely on predefined programs or external modules for control, limiting their adaptability in dynamic or complex environments.

Vision-Language Models for Localization: *PIVOT* (Nasiriany et al., 2024) addresses enabling VLMs to localize actionable points without fine-tuning on task-specific data. Their approach centers on localization through visual question answering, with minimal focus on planning—similar to the role of our Grounding Team. Unlike our method, which integrates both localization and planning within a multi-agent framework, PIVOT primarily addresses localization without managing complex, long-horizon tasks. In PIVOT, a single agent iteratively selects action points, whereas our approach employs multiple agents with distinct roles for refining and verifying actions. A detailed comparison is provided in Appendix E.2.

Language Models as Zero-Shot Trajectory Generators: Kwon et al. (2024) propose using language models as zero-shot trajectory generators. Their approach uses a predefined object detection model (LangSAM) to extract object information, which is then used by the LLM to plan. Specifically, the LLM generates Python scripts to create a trajectory for execution. Unlike our method, which uses a VLLM to integrate perception and action without external modules, their approach relies on separate perception models and code generation for trajectory planning. Further comparison is available in Appendix E.3.

Natural Language as Policies: Concurrent with our work, *Natural Language as Policies (NLaP)* (Mikami et al., 2024) developed a few-shot, end-to-end model for coordinate-level action prediction. Their approach involves providing a one-shot example, either from the same task or a closely related one, rather than adopting a zero-shot paradigm. Unlike our method, which integrates both grounding and planning within a multi-agent framework, NLaP focuses less on grounding and directly uses system information from the environment, bypassing the need to extract coordinates from images using VLMs. NLaP serves as one of the baselines in our experiments, and a detailed comparison is presented in Appendix E.1.

Our Contribution in Context: Our work differs from prior approaches by proposing a zero-shot, single-model, multi-agent system that integrates high-level planning and low-level action execution within a unified VLLM framework. By eliminating the need for external vision encoders and predefined action modules, our method achieves greater adaptability and precision in dynamic environments.

5 EXPERIMENTAL RESULTS

In this section, we evaluate the performance of Wonderful Team across a diverse set of tasks that challenge various aspects of robotic reasoning and manipulation. We address key elements of robotics, including multimodal reasoning, contextual decision-making, and complex spatial planning. Our experiments are categorized into three main groups, each designed to tackle specific challenges while contributing to the broader evaluation of the system's capabilities.

1) Multimodal Reasoning (17 Tasks in Simulated VIMABench)

2) Implicit Goal Inference (3 Custom Real-world Tasks)

3) Spatial Planning (4 Real-world Tasks Adapted from Trajectory Generator)

5.1 MULTIMODAL REASONING - SIMULATED VIMABENCH

To assess our approach's ability to understand multimodal prompts, reason through abstract concepts, and follow constraints, we tested it on all 17 tasks from VIMABench (Jiang et al., 2023). Unlike traditional robotics benchmarks, VIMABench offers a broad range of objects and task types (see Figure 6), requiring advanced scene understanding, multimodal comprehension, and precise planning for manipulation.

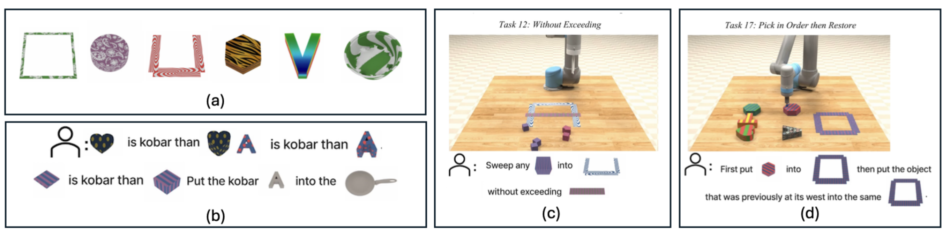


Figure 6: Key Challenges in VIMABench (Jiang et al., 2023): (a) Manipulating uncommon objects and textures, (b) Interpreting multimodal prompts with abstract nouns and adjectives, (c) Executing constraint satisfaction tasks, and (d) Handling Spatial Relations and Sequential Dependencies.

We evaluated all 17 tasks in VIMABench, categorized into four main task suites as defined by Jiang et al. (2023), each targeting distinct robotic capabilities:

1) Simple Object Manipulation: pick-and-place and rotate tasks using multimodal prompts that combine images and text.

2) Novel Concept Grounding: Tasks with abstract terms like "kobar" (see Figure 6(b)), testing the agent's ability to understand and act on novel concepts.

3) Visual Constraint Satisfaction: Manipulating objects while adhering to specific constraints not easily segmentable, such as avoiding certain areas (see Figure 6(c)).

4) Visual Reasoning: Higher-level reasoning tasks that involve understanding object properties and maintaining state, such as "put the object that was previously at its west ..." (see Figure 6(d)).

5.2 IMPLICIT GOAL INFERENCE - REAL ROBOTS

To evaluate our framework's reasoning abilities and visual context understanding in real-world settings, we designed a set of **Implicit Goal Inference Tasks**, each with four variations, to assess the system's capacity for long-horizon reasoning and context-aware high-level instructions interpretation (see Figure 7).

We evaluated our method on three real-world tasks:

1) Fruit Placement: The robot is asked to place each fruit in a color-matched area across various setups using the same general prompt. This task challenges the system to infer the desired placement and sometimes also to identify and correct any initially misplaced fruits (see Figure 7(a)).

2) Superhero Companions: The robot is tasked with placing fruits and snacks based on color similarity, requiring it to identify objects and make suitable matches, even with non-exact color matches, multi-colored objects, and cases where no clear match is available. (see Figure 7(b)).

3) Fruit Price Ranking: The robot is tasked with ranking fruits by price. This challenges the system to interpret visual discount information, apply comparative reasoning, and execute precise ranking to correctly order the fruits (see Figure 7(c)).

All tasks require the system to interpret high-level prompts, perform contextual reasoning, and execute multi-step actions to achieve the implicit goal state based on the provided instructions.

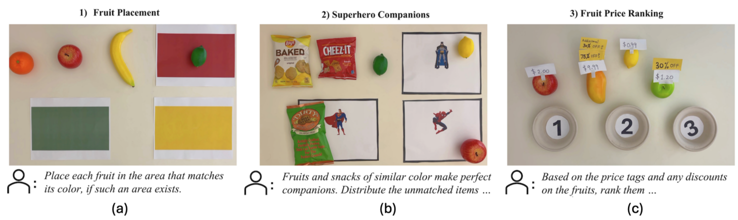


Figure 7: Examples of Ambiguous Instruction & Contextual Reasoning Tasks: (a) Fruit Placement, (b) Superhero Companions, and (c) Fruit Price Ranking.

5.3 Spatial Planning - Real Robots

To further challenge our system, we introduced tasks that require precise planning and subgoal management. These tasks test the agent's ability to produce accurate action sequences and handle dependencies carefully. (see Figure 8).

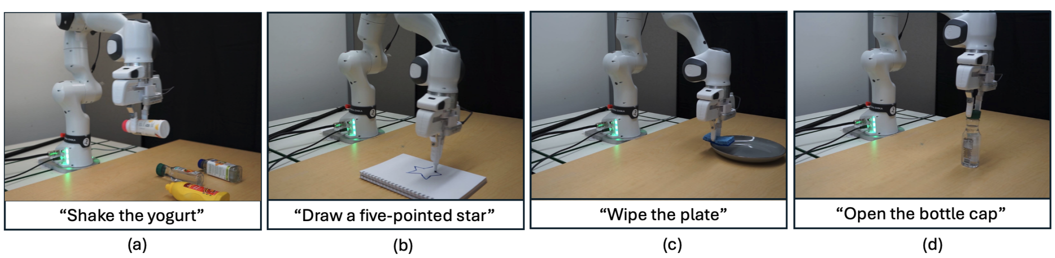


Figure 8: Examples of Complex Planning Tasks.

We evaluated our method on four real-world tasks:

1) Shaking the Bottle: The agent grasps a bottle, shakes it in the air, and places it back on the table. (see Figure 8(a)).

2) Drawing a Five-Pointed Star: The agent holds a marker and draws a five-pointed star on a notebook. This task demands very precise path planning for both lowering the marker to the paper and accurately tracing the star's points (see Figure 8(b)).

3) Wiping the Plate with Sponge: The agent cleans a plate using a sponge. This task involves coordinating the sponge's movement to cover the entire surface of the plate (see Figure 8(c)).

4) Opening a Bottle Cap: The agent grasps a bottle and unscrews its cap (see Figure 8(d)).

All four tasks require the robot to generate accurate intermediate subgoals, carefully plan and execute actions within spatial contexts.

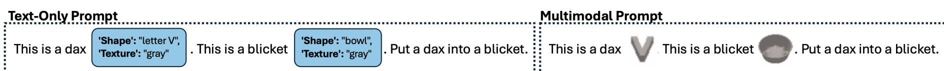
5.4 Results and Discussion

In VIMABench (Jiang et al., 2023), we **compared Wonderful Team** against the following methods: **(1) Trajectory Generator**(Kwon et al., 2024), which uses an LLM for planning and LangSAM for perception; **(2) Natural Language as Policies (NLaP)**(Mikami et al., 2024), which employs one-shot prompting and directly accesses ground-truth coordinates, bypassing perception; and **(3) Ablations Replacing the Grounding Team**, where we replace the multi-agent Grounding Team with a single VLLM for inferring object coordinates directly and a separate vision-language model, OWL-ViT.

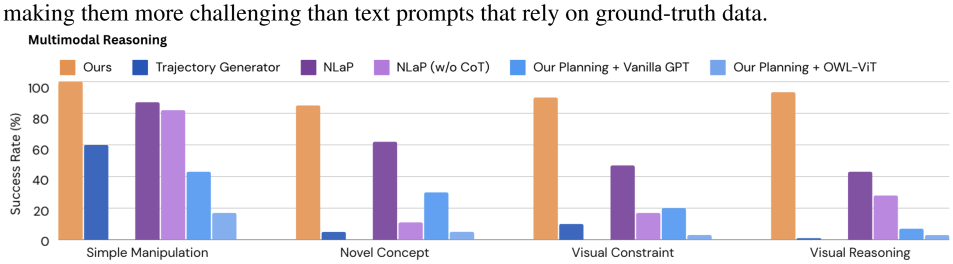
Table 9(a) outlines each method's characteristics, including zero-shot versus one-shot settings, prompt types, and the modules used for planning and perception. Methods without vision rely on text prompts rather than the more complex multimodal prompts (Figure 9(b)). Notably, **NLaP employs one-shot examples** in its prompting and **directly uses the ground truth state coordinates** from the environment, entirely bypassing the perception challenge and, therefore, any comparisons must be made carefully. Due to this lack of perception capability, we can only compare with NLaP in the simulated tasks.

Method	Experience	Planning	Prompt Format	Perception
Ours	Zero-Shot	VLLM	Multimodal *(text + image)*	Multi-Agent VLLM *(GPT)*
Trajectory Generator	Zero-Shot	LLM	Text-Only	VLM *(LangSAM)*
NLaP Variants	One-Shot	LLM	Text-Only	Ground Truth State
Our Planning + Vanilla GPT	Zero-Shot	VLLM	Multimodal *(text + image)*	Single VLLM *(GPT)*
Our Planning + OWL-ViT	VLLM	VLLM	Multimodal *(text + image)*	VLM *(OWL-ViT)*

(a) Comparison with baseline methods. Grey boxes indicate reduced complexity due to the framework's nature, which should be considered when interpreting results.



(b) Examples of prompts: text vs. multimodal. Multimodal prompts require visual understanding, making them more challenging than text prompts that rely on ground-truth data.



(c) Performance on VIMABench tasks. **Wonderful Team** achieves strong results across all task domains. Performance declines when the Grounding Team is removed or replaced.

Figure 9: Overall comparison and results on VIMABench tasks.

As shown in Figure 9(c), Wonderful Team outperforms baselines across all VIMABench tasks. The Grounding Team and multi-agent structure are crucial; removing or replacing them significantly reduces performance. Methods like Trajectory Generator and our ablation with a separate VLM struggle to detect uncommon objects and lack nuanced reasoning for detection and manipulation. Even with perfect localization (as in NLaP), complex long-horizon planning remains challenging without the multi-agent structure, leading to misinterpretations and errors (Appendix E.1). Ablation studies (Appendix C) **confirm the importance of each component in Wonderful Team**.

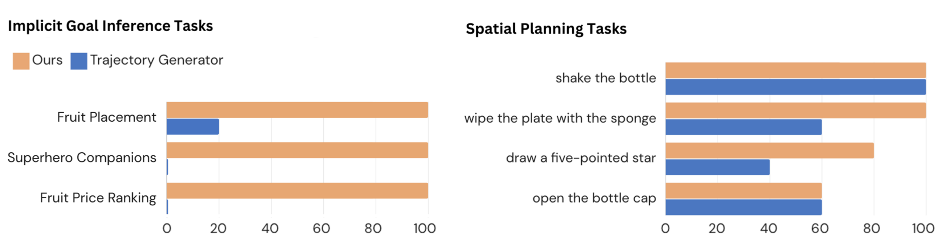


Figure 10: Success rates of Wonderful Team and Trajectory Generator on real-world tasks involving ambiguous instruction tasks and spatial planning tasks.

Implicit Goal Inference Tasks In real robot tasks with more general instructions (e.g., placing fruits based on color), as shown in Figure 10, Wonderful Team achieved a 100% success rate, while Trajectory Generator significantly struggled due to its separation of reasoning and vision. Trajectory Generator relies on an LLM to extract information from the text prompt, which requires explicit instructions. When multiple objects from the same category (e.g., various fruits) were present without specific identifiers, it failed to distinguish between them. Using only "fruit" as the identifier for LangSAM, it could extract the coordinates of all fruits but could not proceed without knowing each fruit's identity and color. Since the LLM lacks grounding knowledge and only has access to these coordinates, it fails to perform meaningful reasoning, resulting in ineffective planning and ultimately causing the low success rate.

Spatial Planning Tasks In real robot spatial planning tasks (e.g., drawing a star), as illustrated in Figure 10, Wonderful Team performed comparably or slightly better, benefiting from the Verification Agent ensuring trajectories were within correct spatial boundaries. The Verification Agent checked the planned paths against workspace constraints (e.g., notebook to draw the star on). Both methods exhibited similar failure modes, often due to depth camera sensor inaccuracies affecting tasks requiring height precision (e.g., particularly problematic for opening a bottle cap). These inaccuracies led to errors in estimating the z-axis position, highlighting areas for future improvement in sensor integration and error correction.

6 Further Discussions

6.1 Comparison with Methods that Train

In recent years, the machine learning community has often seen new LLMs exceed the performance of previous generation fine-tuned models in zero-shot settings, despite the latter's advantage of task-specific tuning. To explore this trend in the context of visual LLMs and robotics, we compare Wonderful Team with several methods that were at least partially fine-tuned on robotics tasks.

In particular, we compare against: 1) VIMA Jiang et al. (2023) and 2) Instruct2Act Huang et al. (2023a). In Table 1, we consistently see that the advantage of fine-tuning loses out to having a more powerful VLLM.

	Ours	**VIMA-200M (L3)**	**Instruct2Act**
Visual Reasoning	Zero-Shot	Domain Fine-Tuned Mask R-CNN	Pre- and Post-Processing
Task Execution	Zero-Shot	BC Offline Learning	Pre-defined API + One-Shot Ex
Success Rate (%)	91.25	88.71	79.67

Table 1: Comparison with non-zero-shot Methods on VIMABench Tasks. Success rates are averaged across the same tasks considered in figure 9(c)

6.2 Limitations: Where does Wonderful Team struggle?

Limited 3D Reasoning and Partial Observability: While the integration of depth cameras allows Wonderful Team to capture 3D data, its reasoning and planning are still largely confined to 2D space. This limitation hinders tasks that require precise manipulation along the height axis or a full understanding of 3D spatial relationships. Additionally, it struggles with partial observability, often leading to incorrect interpretations of spatial relationships.

Real-Time Adaptation and Error Recovery: Although the Replanning Agent is designed to address failures post-execution, the framework could be improved with real-time dynamic error detection to catch issues immediately. However, reprocessing parts of or the entire task can be computationally expensive and sometimes impractical, requiring careful system design. This limitation is particularly important in navigation tasks or rapidly changing dynmaic environments, where constant replanning can be costly and reduce applicability. Improving the system's robustness to environmental variations and enhancing real-time error recovery remain key areas for future work.

REFERENCES

Zhenshan Bing, Alexander Koch, Xiangtong Yao, Kai Huang, and Alois Knoll. Meta-reinforcement learning via language instructions. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5985–5991. IEEE, 2023.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL `https://arxiv.org/abs/2005.14165`.

Arthur Bucker, Luis Figueredo, Sami Haddadin, Ashish Kapoor, Shuang Ma, Sai Vemprala, and Rogerio Bonatti. Latte: Language trajectory transformer. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7287–7294. IEEE, 2023.

Qizhe Gu, Arjun Kuwajerwala, Sean Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aayush Agarwal, Christopher Rivera, Wenzhen Paul, Kyle Ellis, Rama Chellappa, Chuang Gan, Celso M de Melo, Joshua B Tenenbaum, Antonio Torralba, Florimond Shkurti, and Liam Paull. Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning. *CoRR*, abs/2309.16650, 2023.

Yingdong Hu, Fanqi Lin, Tong Zhang, Li Yi, and Yang Gao. Look before you leap: Unveiling the power of gpt-4v in robotic vision-language planning, 2023.

Siyuan Huang, Zhengkai Jiang, Hao Dong, Yu Qiao, Peng Gao, and Hongsheng Li. Instruct2act: Mapping multi-modality instructions to robotic actions with large language model. *arXiv preprint arXiv:2305.11176*, 2023a.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *Proceedings of Machine Learning Research*, 162:9118–9147, 2022a.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *CoRR*, abs/2201.07207, 2022b. URL `https://arxiv.org/abs/2201.07207`.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022c.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023b.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, and et al. Do as i can, not as i say: Grounding language in robotic affordances. *Proceedings of Machine Learning Research*, 205:287–318, 2022.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts, 2023.

Teyun Kwon, Norman Di Palo, and Edward Johns. Language models as zero-shot trajectory generators. *IEEE Robotics and Automation Letters*, 9(7):6728–6735, 2024. doi: 10.1109/LRA.2024.3410155.

J Liang, W Huang, F Xia, P Xu, K Hausman, B Ichter, P Florence, and A Zeng. Code as policies: Language model programs for embodied control. *ICRA*, pp. 9493–9500, 2023.

Oier Mees, Jessica Borja-Diaz, and Wolfram Burgard. Grounding language with visual affordances over unstructured data, 2023.

Yusuke Mikami, Andrew Melnik, Jun Miura, and Ville Hautamäki. Natural language as policies: Reasoning for coordinate-level embodied control with llms, 2024.

Soroush Nasiriany, Fei Xia, Wenhao Yu, Ted Xiao, Jacky Liang, Ishita Dasgupta, Annie Xie, Danny Driess, Ayzaan Wahid, Zhuo Xu, Quan Vuong, Tingnan Zhang, Tsang-Wei Edward Lee, Kuang-Huei Lee, Peng Xu, Sean Kirmani, Yuke Zhu, Andy Zeng, Karol Hausman, Nicolas Heess, Chelsea Finn, Sergey Levine, and Brian Ichter. PIVOT: Iterative visual prompting elicits actionable knowledge for VLMs. In *Forty-first International Conference on Machine Learning*, 2024. URL `https://openreview.net/forum?id=051jaf8MQy`.

Ishan Singh, V Blukis, A Mousavian, A Goyal, D Xu, J Tremblay, D Fox, J Thomason, and A Garg. Progprompt: Generating situated robot task plans using large language models. *ICRA*, pp. 11523–11530, 2023.

Austin Stone, Ted Xiao, Yao Lu, Keerthana Gopalakrishnan, Kuang-Huei Lee, Quan Vuong, Paul Wohlhart, Brianna Zitkovich, Fei Xia, Chelsea Finn, et al. Open-world object manipulation using pre-trained vision-language models. *arXiv preprint arXiv:2303.00905*, 2023.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

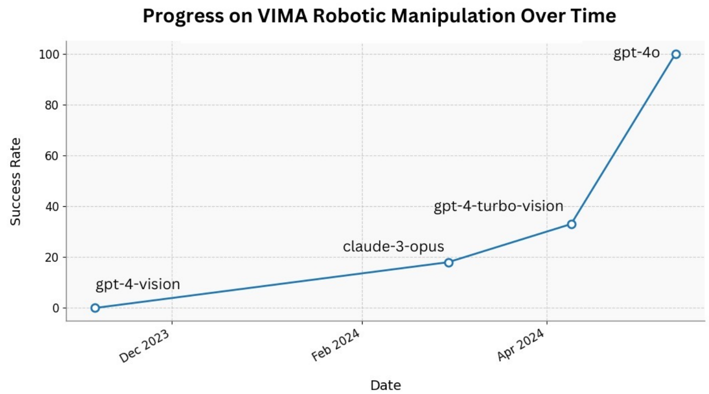
S Vemprala, R Bonatti, A Bucker, and A Kapoor. Chatgpt for robotics: Design principles and model abilities. *CoRR*, abs/2306.17582, 2023.

Andy Zeng, M Attarian, B Ichter, KM Choromanski, A Wong, S Welker, F Tombari, A Purohit, MS Ryoo, V Sindhwani, J Lee, V Vanhoucke, and P Florence. Socratic models: Composing zero-shot multimodal reasoning with language. *ICLR*, 2023.

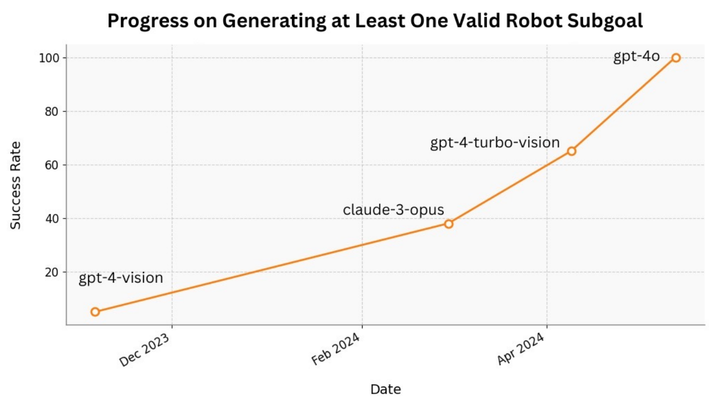
Appendix Table of Contents

A THINGS ARE MOVING EXTREMELY FAST

While it is readily apparent to everyone that LLM progress has been rapid since 2021, it is perhaps less apparent how rapidly these capabilities are influencing robotics. The initial version of this project, which was started in 2022, was largely dead in the water, because VLLMs at the time struggled greatly to understand their environment. In the past year, VLLMs have improved rapidly, which has allowed them to make substantial progress on robotics environments. To better understand this progress, we took and changed the language model to earlier VLLMs. The results roughly track the average performance our system has been able to obtain over time.



(a) Improvement of VLLMs on robotics tasks over time.



(b) Ability of VLLMs to generate at least one valid subgoal.

Figure 11: Progress of VLLMs in robotics, presenting the success rates evaluated on VIMABench tasks, the same benchmarks used in Figure 9(c), highlighting the impact of each modification.

As we can see, the capabilities of these underlying vision-language models are improving at a blistering pace. Suppose we instead consider a slightly easier problem: the ability of with VLLMs to generate at least one valid subgoal, which shows the system is working to some extent but perhaps lacks more refined planning ability. In Figure 11(b), we see that here too the improvements have been rapid.

In the Appendix D, we examine the impact of this rapid progress on the grounding team in particular, and show that older VLLMs often struggled to draw bounding boxes with any regularity, suggesting they lacked the fidelity needed for fine-grained robotic control.

B Experimental Details

B.1 Evaluation Protocol

All experiments were conducted with consistency and rigor to accurately assess our framework's performance.

- **Multimodal Reasoning & Constraint Manipulation**: Each task was executed in 10 runs, allowing only a single attempt per run. An open-loop, single-attempt evaluation protocol was employed to ensure fair comparisons with existing methods and to effectively evaluate the capabilities of the multi-agent framework.
- **Ambiguous Instruction Contextual Reasoning**: Each task was performed in 2 runs for each of the 4 variations with varying difficulty. For instance, increasing the number of price tags for fruit ranking. An open-loop, single-attempt evaluation protocol was used to consistently measure the system's ability to interpret and execute ambiguous instructions.
- **Spatial Planning & Execution**: Each task was carried out in 5 runs under a closed-loop evaluation protocol, permitting up to three replanning attempts. This method assesses the system's ability to manage complex planning, handle unforeseen challenges, and execute multi-step procedures with precision and coordination.

B.2 Multimodal Reasoning - Simulated VIMABench

VIMABench features 17 tabletop manipulation tasks, including pick-and-place and push, with various combinations of objects, textures, and initial configurations. It includes 29 objects with 17 RGB colors and 65 image textures, many of which are uncommon in other robotics tasks, making them ideal for testing our approach. We selected VIMABench because it presents a significant variety of objects and textures compared to traditional environments with easily detectable items. This requires advanced scene understanding and careful planning for successful manipulation. VIMABench also includes multimodal prompts with images and textual instructions, creating a complex and realistic testing environment that necessitates reasoning and long-horizon planning.

B.2.1 Task Details

Simple Object Manipulation: Tasks such as "put ⟨object⟩ into ⟨container⟩," where each prompt image corresponds to a single object. These tasks test the basic pick-and-place capabilities of the system.

Novel Concept Grounding: Tasks with abstract terms like "fax" and "blicket" paired with images, testing the agent's ability to internalize and act upon newly introduced concepts quickly.

Visual Constraint Satisfaction: Tasks that require the robot to perform actions like pushing objects while adhering to specific constraints, such as not exceeding certain boundaries or avoiding designated areas. These tasks test the system's safety and precision in manipulation.

Visual Reasoning: Tasks involving higher-level reasoning skills, such as "move all objects with the same textures into ⟨location⟩," and visual memory tasks like "put ⟨object⟩ in ⟨location⟩ and then restore them to their original position." These tasks assess the framework's ability to reason about object properties and maintain state over multiple actions.

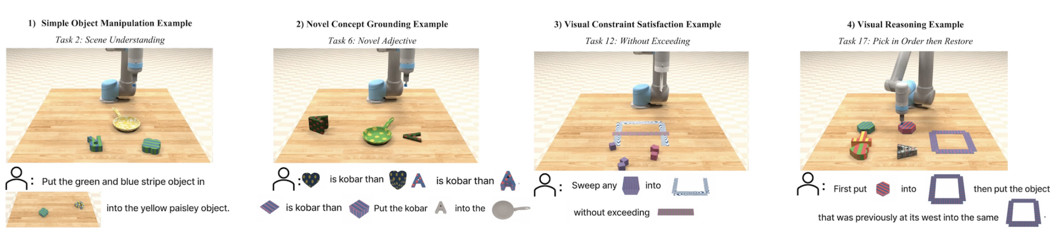


Figure 12: Examples of tasks in VIMAbench Tasks(Jiang et al., 2023).

B.2.2 Full Experimental Results

In the main paper, we presented results from a selective number of tasks within four categories out of the 17 VIMABench tasks. This was due to the nature of some tasks not being optimal for visual testing. For instance, the twist task requires the robot to determine the precise degree of rotation from before and after images, a challenge without prior training on such tasks.

In Table 2, we present the full experimental results across all 17 tasks of VIMABench. VIMABench defines six main categories of tasks, which are separated in the table by alternating grey and white blocks. From top to bottom, these categories are: Simple Object Manipulation, Visual Goal Reaching, Novel Concept Grounding, One-shot Video Imitation, Visual Constraint Satisfaction, and Visual Reasoning.

Table 2: Success Rates Across All VIMABench Tasks

Task Num	VIMA 200M	Instruct2Act	NLaP (w/o CoT)	NLaP	TG	Ours
1: Visual Manipulation	99	91	93	100	60	100
2: Scene Understanding	100	81	60	67	40	100
3: Rotate	100	98	93	93	80	100
*4: Rearrange	97	79	52	73	-	80
*5: Rearrange then Restore	54.5	72	25	73	-	70
6: Novel Adjective	100	82	13	43	10	70
7: Novel Noun	99	88	8	80	0	100
*8: Novel Adjective and Noun	-	-	-	-	-	60
*9: Twist	17.5	-	-	-	-	50
*10: Follow Motion	-	35	0	12	-	10
*11: Follow Order	90.5	72	0	0	-	0
12: Without Exceeding	93	68	17	47	10	90
*13: Without Touching	-	0	0	3	-	40
*14: Same Texture	-	80	3	71	-	100
15: Same Shape	97.5	78	10	80	0	100
16: Manipulate Old Neighbor	46	64	8	20	0	90
17: Pick in Order then Restore	43.5	85	10	30	0	90

Tasks marked with a star were excluded from the main paper's results for the following reasons: **1. Nature of Tasks:** Categories Visual Goal Reaching (Task 4 and 5) and One-shot Video Imitation (Task 10 and 11) were excluded because these tasks are not the best indicators of VLLM's capabilities without additional prompting.

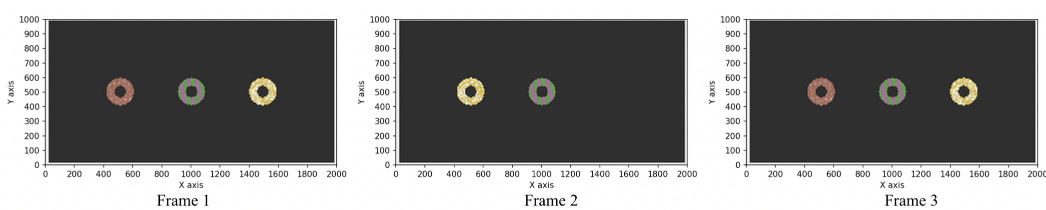


Figure 13: Comparison between images without and with ticks for positional reference.

For example, as shown in Figure 13, Task 11 in the One-shot Video Imitation category requires examining several consecutive frames as ’goal scenes’. Without further task-specific prompting or training, it is very challenging to infer the required actions between frames since there isn’t a single correct answer. For instance, transitioning from Frame 1 to Frame 2 in this example could be achieved by moving the yellow O onto the red O, or by first removing the red O and then moving the yellow O to the same position. By nature, these tasks require additional tools or workflows, which complicate zero-shot evaluation. Additional prompting on tasks like this to help the VLLMs better understand the relationship between frames will probably be helpful. However, this is not the focus of our research, so we used the same prompt for these evaluations in Table 2.

2. Missing Baseline Results: Tasks 8, 9, 13, and 14 were excluded due to the lack of available baseline results for comparison.

A complete list of tasks with video illustrations can be found here.

B.3 Implicit Goal Inference - Real Robots

B.3.1 Task Details

As discussed in Section 5, we evaluated our method on three real-world tasks. This section provides more examples of the diverse scenes used for each task.

Fruit Placement: The robot is given a random set of fruits and areas of different colors. The prompt is:

> “Place each fruit in the area that matches its color, if such an area exists.”

Some scenarios included fruits with no matching color or mismatched colors.

Superhero Companions: The robot is provided with fruits and snacks of different colors and three bins designated for different superheroes. The prompt is:

> “Fruits and snacks of similar color make perfect companions. Distribute the unmatched items from the top left corner to the superheroes to help each of them have companion pairs.”

Fruit Price Ranking: Various fruits with price tags are presented to the robot. The prompt is:

> “Based on the price tags and any discounts on the fruits, rank them from the most expensive to the cheapest and place them in the corresponding bowl.”

To further challenge its visual and reasoning skills, we added promotional discounts on top of the original price tags.

(a) Fruit Placement (b) Superhero Companions (c) Fruit Price Ranking

Figure 14: Examples of task environments: (a) Fruit Placement, (b) Superhero Companions, (c) Fruit Price Ranking.

B.3.2 Robot Setup

For our real-world experiments, we used the UFactory xArm 7, a versatile robotic arm with 7 degrees of freedom, a maximum payload of 3.5 kg, and a reach of 700 mm. It was controlled via the xArm Controller using Python and ROS, allowing seamless integration with our multi-agent system. The robot was equipped with a 2-finger gripper for manipulating various objects. The experiments were conducted on a standard laboratory workbench with predefined task areas, and the robot was calibrated before each experiment to ensure accurate positioning and movement. Our framework mapped the relative displacement of the target position to the robot arm and the pixel coordinates used by the framework, enabling precise picking and placing actions.

For the visual input, we set up a camera directly above the predefined task area, as the robot itself does not come equipped with one. This setup provided a clear and consistent view of the workspace, allowing the VLLM to interpret the environment accurately and plan actions effectively.

B.3.3 RESULTS

Our real robot experiments demonstrated that our framework successfully completed all three tasks 100% of the time. Note that we **did not modify any of the prompt or pipeline** moving from simulated VIMABench environment to the tasks on the real robot. It was surprising to us how robust the reasoning and planning capabilities of are. This section provides qualitative results from these experiments, illustrated in Figures 15, 16, and 17. These figures highlight specific aspects of the tasks, illustrating the effectiveness of our framework. It is important to note that these results only reflect the work of the planning team. The role of the grounding team, locating objects and determining their positions, is crucial for the successful execution of these plans.

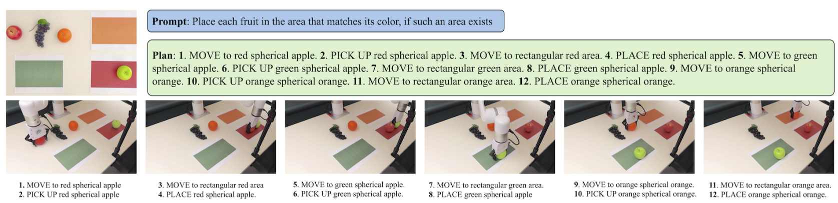


Figure 15: Example Execution on Fruit Placement Task

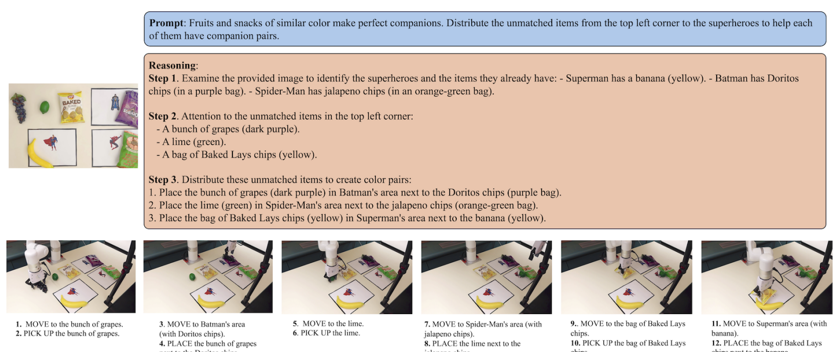


Figure 16: Example Execution on Superhero Companions Task

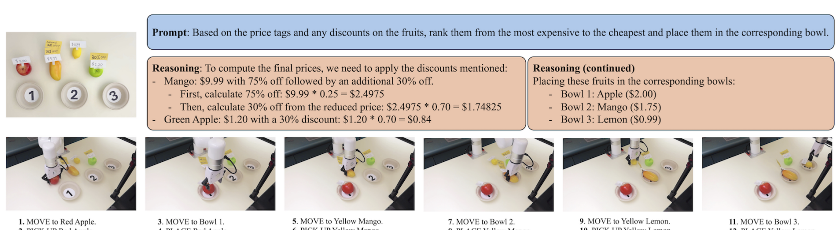


Figure 17: Example Execution on Fruit Price Ranking Task

In the fruit placement task (Figure 15), we present the final execution plan to illustrate the structure of a complete plan. Due to the straightforward nature of the task, this figure does not include the reasoning process. For the superhero companions and fruit price ranking tasks (Figures 16 and 17), we emphasize the reasoning process and omit the block for the complete final plan for the sake of conciseness. The final plans for these tasks are similar in structure to the fruit placement task, essentially combining the substeps in the execution sequence at the bottom of the figures.

Videos of the experiments and actual execution can be viewed here.

B.4 Spatial Planning - Real Robots

B.4.1 Robot Setup

For our real-world experiments, we used the Franka Emika Panda robot, a 7-degree-of-freedom robotic arm controlled using ROS. We used an Intel RealSense D435 camera positioned above the workspace to extract visual and depth information.

For top-view D-RGB images, the camera was mounted directly above the predefined task area, as the robot itself does not come equipped with an onboard camera. This setup provided a clear and consistent view of the workspace, allowing the VLLM to accurately interpret spatial relationships and plan actions. The depth information was especially valuable for tasks that required accurate height estimation and object manipulation.

C ABLATION STUDIES: ARE ALL PARTS OF NECESSARY?

In this section, we present an ablation study to isolate and evaluate the contributions of our proposed hierarchical prompting mechanism relative to the capabilities of gpt-4o itself. The objective is to determine the extent to which the hierarchical prompting enhances system performance beyond what gpt-4o alone can achieve.

We systematically remove or modify various components of our system, such as the Verification Agent and the Box Checking Agent, to observe their individual impacts on performance. This process helps to identify the specific contributions of each component within the hierarchical framework.

The study addresses the following key questions:

- How significant is the hierarchical prompting mechanism in improving system performance compared to gpt-4o alone?
- What are the individual contributions of the agents to the system's accuracy and efficiency?
- How does the removal or modification of these components affect performance metrics?

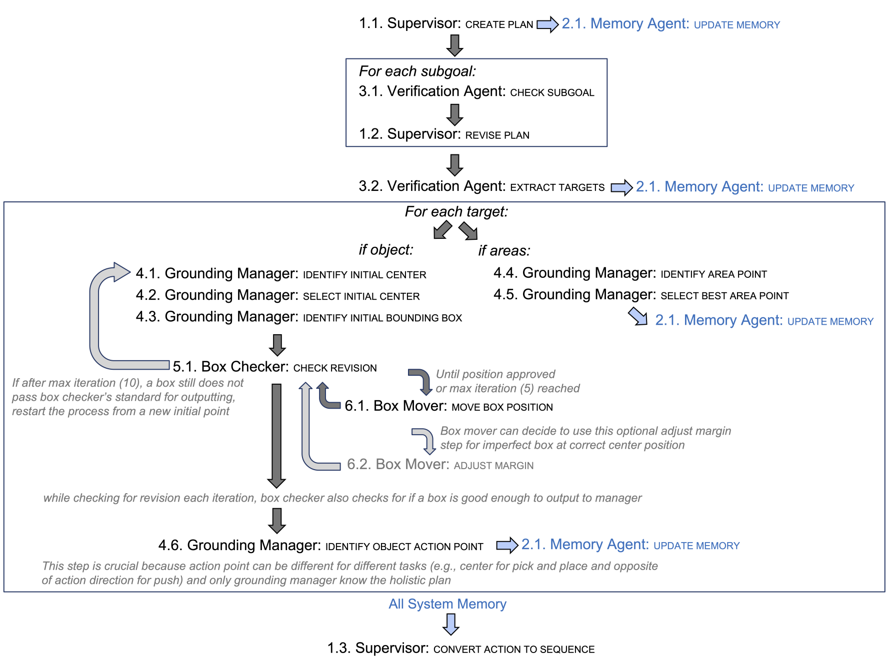


Figure 18: Workflow: Complete

Figure 18 shows the workflow of the complete framework of . We also provide the full prompt and example input and output corresponding to this workflow chart in Appendix C for more concrete details.

We systematically removed or modified various components of our system, such as the Verification Agent and the Box Checking Agent, to observe their individual impacts on performance. This approach helps identify the specific contributions of each component within the hierarchical framework.

The study addresses the following key questions:

- How significant is the hierarchical prompting mechanism in improving system performance compared to GPT-4o alone?
- What are the individual contributions of the agents to the system's accuracy and efficiency?

- How does the removal or modification of these components affect performance metrics?

Figure 18 shows the workflow of the complete framework of . Detailed prompts, input examples, and output corresponding to this workflow can be found in Appendix C.

To isolate the effects, we tested the following configurations:

- **1: Removing the Verification Agent:** Without the Verification Agent, the system directly used the supervisor's initial set of subgoals as the final output. This led to errors, as there was no reflection to refine subgoals based on real-time feedback.
- **2: Removing the Box Checking Agent:** The Box Checking Agent evaluates proposed revisions by the Box Mover for improvements and final output quality. When removed, the Box Mover had to perform self-checks, resulting in less accurate outcomes due to the lack of a secondary verification layer.
- **3: Removing Both the Verification and Box Moving Agents:** The system relied solely on the initial bounding box identified by the Grounding Manager, skipping the iterative refinement process and leading to suboptimal action points.
- **4: Removing the Box Checking Agent and Box Moving Agent:** The initial grounding position was used directly without any further verification or adjustments, significantly affecting the robot's ability to select precise action points.
- **5: Removing the Verification Agent, Box Checking Agent, and Box Moving Agent:** The supervisor operated independently, approximating coordinates directly from the image without hierarchical feedback or bounding box identification, resulting in reduced accuracy and adaptability in task execution.
- **6: Removing the Grounding Team:** The supervisor generated plans and extracted targets without identifying bounding boxes, leading to a decline in precision for coordinate-level actions.
- **7: Removing the Verification Agent and Grounding Team:** The supervisor handled all steps, from planning to coordinate generation. Without the Grounding Team, the system relied on rough estimations for actionable points, reducing overall accuracy.
- **8: Removing the Memory Agent:** The Memory Agent selectively stores important information to reduce hallucinations and aid in complex, long-horizon tasks. Its removal had a lesser impact on simpler tasks but proved crucial for maintaining key information in more complex scenarios involving multiple subgoals.

In summary, our settings considered can be summarized in Table 3.

Table 3: Settings Summary

Setting Number	Supervisor	Verification	(G) Manager	(G) Checker	(G) Mover	Memory
1	✓	✗	✓	✓	✓	✓
2	✓	✓	✓	✗	✓	✓
3	✓	✗	✓	✗	✓	✓
4	✓	✓	✓	✗	✗	✓
5	✓	✗	✓	✗	✗	✓
6	✓	✓	✗	✗	✗	✓
7	✓	✗	✗	✗	✗	✓
8	✓	✓	✓	✓	✓	✗

Table 4 shows the results of the main tasks from the four primary task suites used in our comparison in Figure 9(c).

Table 4: Success Rates Across Different Settings

Task Num	Complete	1	2	3	4	5	6	7	8
1: Visual Manipulation	100	100	80	80	60	50	50	70	100
2: Scene Understanding	100	70	60	60	60	70	60	20	100
3: Rotate	100	60	80	60	70	30	40	80	100
6: Novel Adjective	70	30	20	0	30	0	10	0	50
7: Novel Noun	100	60	80	60	40	20	20	20	70
12: Without Exceeding	90	10	20	10	0	0	10	10	40
15: Same Shape	100	10	10	10	0	0	0	20	60
16: Manipulate Old Neighbor	90	30	40	20	10	0	10	0	50
17: Pick in Order then Restore	90	0	0	0	0	0	0	0	40

Generally speaking, tasks with higher task numbers are typically more complex, involving longer horizons and requiring more sophisticated reasoning. The verification and memory agents are particularly beneficial in complex environments with multiple subgoals. Removing them from the framework often results in failure modes such as treating irrelevant distractor objects as task objects or misidentifying arbitrary empty spaces as target locations.

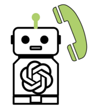
Omitting grounding members tends to lead to less accurate coordinates, which can impact performance. Even for simple tasks without long-horizon planning, the lack of precise grounding can hinder task execution and result in suboptimal outcomes.

Interestingly, the simplest version, where only a supervisor is used, achieved decent success rates on simpler tasks. This could be due to the framework's reduced complexity with fewer components. Simpler tasks usually involve only two or three task objects and locations, making them manageable by the supervisor. There is also a higher probability of guessing an actionable location for larger objects. However, failure modes in this setting include the lack of precise location identification and partially incorrect or infeasible plan. When tasks become more complicated, the absence of corrective processes often leads to failure, especially when hallucination is common.

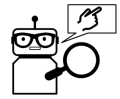
C.1 Understanding What Each Part of Wonderful Team Does

Below, we give a summary of this section, summarizing the responsibilities of each team member and how the overall system suffers if we remove them. This shows the relative strength of the multi-agent approach, and how when working together the team members can compliment each other's strengths.

supervisor

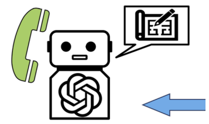


Responsibility	Receive the initial task, develop a plan for carrying out the task including subgoals. Verify the plan is followed and send the final actions to the robot.
Prompt	You have received a multimodal robotic task description in the form of a combination of text and images, followed by a top-view and a front-view image of the environment. Your task is to interpret this combination of text and images and output a plan with key subgoals.….[more details about environment and specific goals]
Input	A textual description of the task and an image of the environment.
Output	A subplan of steps that should be followed to achieve a goal. After the subplan is executed, this agent returns the final actions the agent should take.
What happens without it?	If we replace the multi-agent framework with a flat single agent structure, success on all tasks in VimaBench fall dramatically. For simple tasks like Visual manipulation, this fall is from 100% to 70%. For complex tasks like “Pick in Order and Restore” success goes from 90% to 0%. Similar results are seen on the real robot. The key advantage of the multi-agent framework is that it can self-correct in sub-loops, protecting against hallucination or bad initial estimation. Single agent methods such as NLaP and PIVOT often struggle with precise object manipulation and visual reasoning.

Grounding Team 

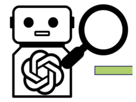
Responsibility	Identify the location of objects in the environment. Tell the robot the correct action points (points where it should center its gripper when interacting with objects)
Prompt	You are an agent that plays a crucial role in a multi-agent robotic system, responsible for accurately identify coordinates of target locations and objects in a robotic environment….[more details about environment and specific goals]
Input	A high-level plan, a top-view images with x and y axis ticks, and a specific object of interest to identify
Output	Thought process. Final (x, y, z) location of object center points.
What happens without it?	The agent can not corre ctly identify the location of objects in the scene, leading to imprecise actions. Consequently, on simple visual manipulation, success falls from 100% to 50%. The grounding team is important because it can iteratively improve upon its estimate of the location of key objects in the environment. Normal VLLM estimates of key points are noisy. But the model is capable of self-correcting initial estimates by looping with the grounding team. This is not possible with a single agent structure.

Memory Agent



Responsibility	Managing a memory dictionary, which has locations of key objects in the environment, and past plan for object manipulations provided by the supervisor.
Prompt	You will receive a system memory dictionary, an agent's name, a response from that agent, and a context of this response generated by the agent itself. Your task is to determine if this information is relevant to successful task execution. If so, summarize and update system memory of this information.
Input	Memory dictionary, output from other agents, context of generated outputs.
Output	Thought process, Updated memory dictionary with locations of key objects from the prompt.
What happens without it?	Tasks such as “pick in order then restore,” rely on memories of previous actions. Without memorizing the order of previous actions, success rates on these tasks fall from 90% to 40%. In general, the performance on most tasks suffer because the agent struggles to remember where it is in task execution. The supervisor becomes burdened trying to remember this information and suffers from hallucinations.

Verification Agent



Responsibility	Analyze the high-level plan provided by the supervisor, paying attention to potential environmental hazards. Especially consider feasability. Ask informative or clarifying questions.
Prompt	You are an agent that plays a crucial role in a multi-agent robotic system, responsible for verifying a given high-level plans with each subgoal for the successful execution of robotic tasks in a specific environment. [more details about environment]
Input	High level plan from the supervisor. Image of the environment.
Output	Either a clarification question or concern related to the feasibility of the generated plan, or approval to execute the plan.
What happens without it?	In "Without Exceeding," if there is no Verification Agent then the supervisor often fails to consider where it must stop the sweeping action. The supervisors instructions are also overly ambiguous about how many objects need to be moved, even though this is explicitly in the task command! If we give the LLM the ability to self-verify with the Verification agent, then success on Without Exceeding increases from 10% to 90% because the agent double checks its ambiguities and corrects them. Similar effects are observed in Scene Understanding and Rotate, where success rises from 70% to 100% and 60% to 100% respectively upon the inclusion of the Verification Agent.

D ABLATION STUDIES: VLLMS' SPATIAL REASONING LIMITATIONS AND POTENTIALS

D.1 EVALUATING VLLM'S SPATIAL UNDERSTANDING

We aim to answer the question: **How capable are VLLMs at finding accurate actionable position coordinates?**

We set up a toy tabletop environment with various colored and shaped objects placed on a grey table mat, with a single target object (a circle) used to calculate deviation. An example of the environment is shown in Figure 19.

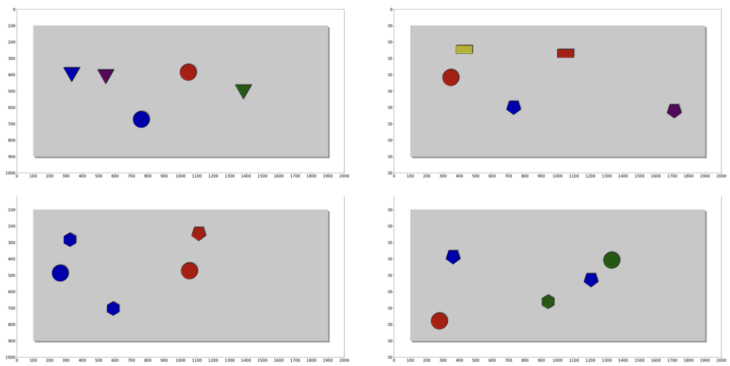


Figure 19: Toy Environment Illustration

We prompt different VLLMs to provide actionable coordinates for the target object, using the overlaid pixel coordinates as a reference. Our goal is to determine whether the coordinates generated by VLLMs are directly usable for action generation and execution.

D.1.1 EXPERIMENTAL SETUP

We tested three state-of-the-art VLLMs:

- **GPT-4o**
- **GPT-4-turbo-vision**
- **Claude-3-opus**

Each model was asked to provide the coordinates of the target object based on the given image with pixel coordinates.

D.1.2 RESULTS

Are the coordinates directly usable? Using this simple environment, we want to answer this question we asked earlier concretely. Although actual robotics environments can look much more complicated visually, we can get an idea of the performance of these models. Any point with deviations from the circle center smaller than the circle radius is considered actionable (lies on the circle for picking).

Table 5: Success Rates of Directly Usable Coordinates

Model	Success Rate (%)
GPT-4o	33
GPT-4-turbo-vision	5
Claude-3-opus	4

We can see from Table 5 that earlier models have a very low success rate. Even with the very strong GPT-4o model, directly using the generated coordinates, even with a perfect plan, can only achieve a 33% success rate, which is far from optimal, not to mention the simple nature of this task.

D.1.3 DEVIATION ANALYSIS

Are the coordinates at least somewhat close to the target objects?

Although the generated coordinates might not be directly usable for action generation, we wondered if the coordinates are at least informative and close to the target objects for further refinements. In the toy environment, we illustrate the circle of 3 times the radius of the original target circle (the radius of the target circle is always 50 here). This seems to be a good definition of being close in the environment. However, we tried different thresholds to see a fuller picture, as shown in Table 6.

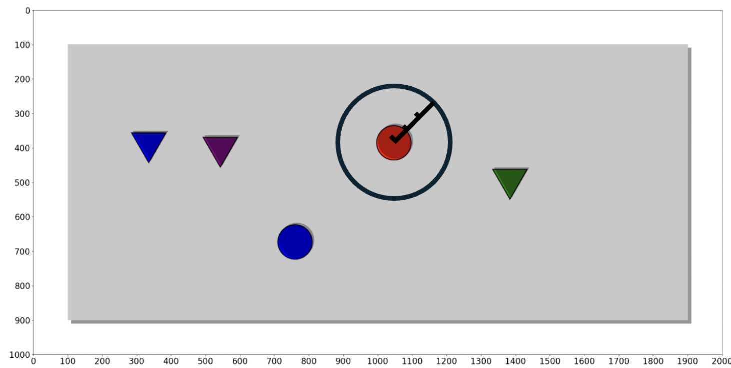


Figure 20: Illustration of the definition of "close to" ($3\times$ radius) target objects.

Table 6: Deviation Analysis of Generated Coordinates

Model	$\leq 3\times$ radius (%)	$\leq 4\times$ radius (%)
GPT-4o	89	97
GPT-4-turbo-vision	46	68
Claude-3-opus	19	58

From the table, we can see that although not directly actionable, the proposed coordinates of GPT-4o are of pretty good quality and can be refined with improvements. They are mostly around the target objects, indicating great potential for further refinement and effective use in real-world tasks.

D.2 EVALUATING VLLMS' ERROR RECOGNITION AND CORRECTION

Given that VLLMs have the power to estimate positions, can we build a framework that can self-improve? A major component needed here is an agent to check or modify the proposed coordinates. In many robotics tasks, the goal of position finding starts with identifying a bounding box around objects. Suppose we have some proposed bounding box for the object of interest. To further improve upon the initial version, VLLMs need to know if a bounding box is good enough, or if it is completely wrong and should restart from generating a new one instead of modifying the current one. The question we ask is: **Are the VLLMs capable of visually examining and evaluating proposed coordinates?**

D.2.1 EXPERIMENTAL SETUP

To test this ability, we randomly generated 4 types of bounding boxes around the circle of interest. Examples are shown in 21. The types are:

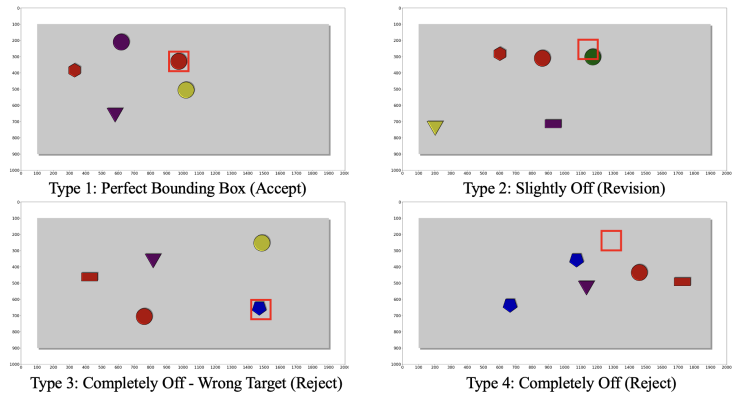


Figure 21: Bounding Box Types: 1) Perfect Bounding Box, 2) Slightly Off, 3) Completely Off - Wrong Target, 4) Completely Off

1. Perfect Bounding Box: The bounding box is correctly placed around the target.

2. Slightly Off: The bounding box is close but not perfectly aligned with the target.

3. Completely Off - Wrong Target: The bounding box is around a different object.

4. Completely Off - Around: The bounding box is sampled around the target (within $4\times$ radius) but is far enough and significantly misplaced, not touching or including the target at all.

Specifically, we give the model a randomly generated bounding box and use the following prompt

> "In the given plot, You are tasked with checking if a bounding box should be accepted, accepted with revision, or rejected.
> Follow these guidelines to determine whether to accept, advise, or reject the new bounding box:
> Criteria:
> - **Accept**: If the bounding box covers the target object well without much extra space, pretty much a perfect bounding box
> - **Revision Needed**: If the bounding box covers at least a small part of the desired object, but more precision is needed
> - **Reject**: If the bounding box is completely irrelevant and does not even touch the desired object
> The target object is: [color] circular object.
> Your output should be in the following text format. Do not include anything else in your output. This means no reasoning process, no json-like format, no explanation, no other types of texts.
> **Output Format:**
> Accept Or
> Revision Needed
> Or
> Reject"

D.2.2 Results

Table 7: Success Rates of Classifying Bounding Boxes

Model	Success Rate (%)
GPT-4o	97
GPT-4-turbo-vision	72
Claude-3-opus	33

From Table 7 and 8, we can see that GPT-4o demonstrated a very strong ability to examine and decide whether a bounding box is good enough just by visual inspection. This capability opens up new possibilities for self-refinements using current VLLMs. Even in cases where initial coordinate generation is not perfect, incorporating a checker as an additional layer of safety along the pipeline can iteratively improve coordinate accuracy until a satisfactory result is achieved.

	gpt-4o			gpt-4-turbo			claude-3-opus		
Ground Truth	Accept	Revision	Reject	Accept	Revision	Reject	Accept	Revision	Reject
Perfect	25	0	0	18	6	1	22	2	1
Slightly Off	1	24	0	0	24	1	19	4	2
Completely Off - Around	0	2	23	0	11	14	23	0	2
Completely Off - Wrong Object	0	0	25	0	9	16	19	1	5
Total	**100**			**100**			**100**		

Table 8: Evaluation of Grounding Box Decisions by GPT-4o, GPT-4-turbo, and Claude-3-Opus Against Ground Truth Across 100 Examples (4 Ground Truth Classes, 25 Examples Each).

In previous tests with Claude-3-opus, the checker often hallucinated during tasks, making it unreliable. For instance, when a bad bounding box is accepted, it not only leads to unsuccessful execution but also confuses the agent itself or other agents in a multi-agent system. This level of complete hallucination is very detrimental. However, in cases where a slightly off bounding box is accepted or a completely off box is sent for revision, it can still be corrected by later parts of the workflow. As shown in Table 8, this level of complete hallucination is predominantly seen in Claude-3-opus outputs. In contrast, the strong performance of GPT-4o suggests that a more reliable approach is now feasible.

E Comparison with Other Methods

E.1 Replicating Natural Language as Policies Using gpt-4o

In Section 5, we presented experimental results of the Natural Language as Policies (NLaP) system as reported in the original paper (Mikami et al., 2024). Their implementation utilized gpt-3.5, whereas our method leverages the more advanced gpt-4o. To ensure a fair comparison, this section presents the results of replicating the NLaP system using gpt-4o.

However, since NLaP does not provide their codebase or the full prompt, including images and object information for the one-shot examples used, we attempted to recreate their framework by writing one-shot examples for each task with human-labeled coordinates and object names according to the framework shown in Figure 1 of their paper. For the one-shot prompt, we closely followed and mimicked their provided prompt examples in Table V.

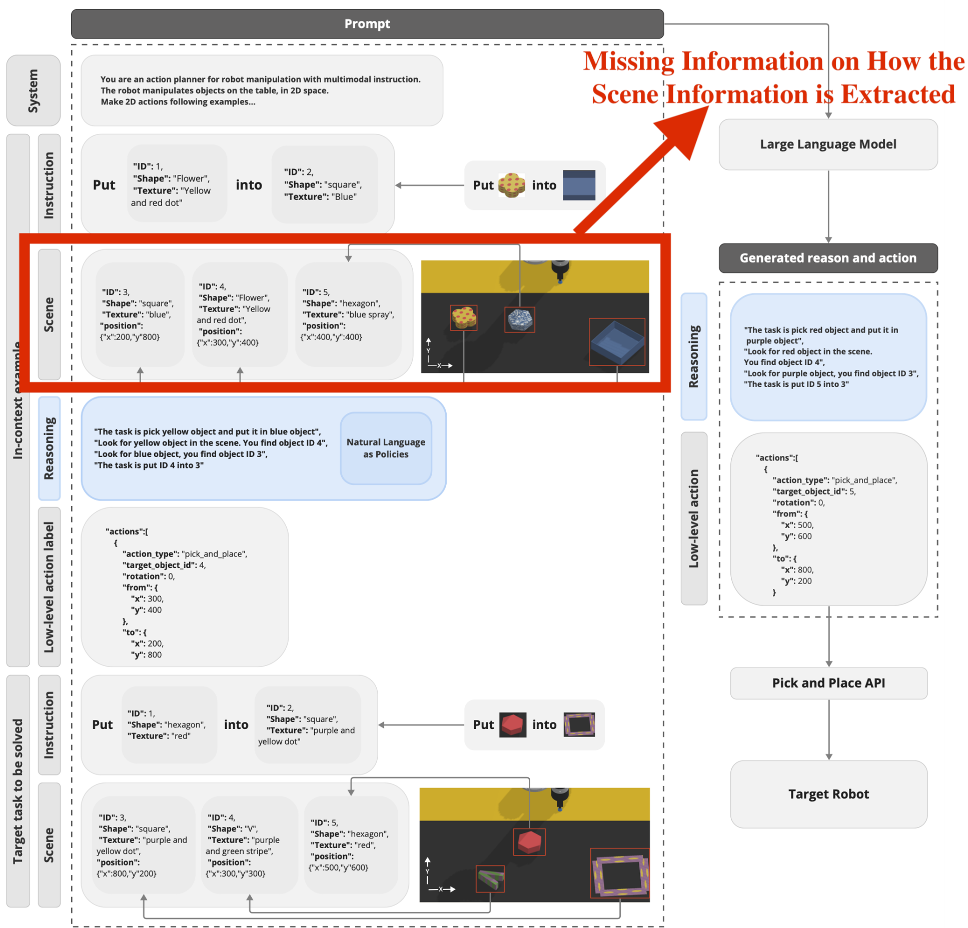


Figure 22: Workflow of Natural Language as Policies by Mikami et al. (2024)

While implementing their framework, we realized that NLaP **does not use the framework to extract coordinate information.** Instead, the extracted coordinates are provided and given to the LLM. The authors did not mention how the coordinates were extracted; the only job of the LLM is to incorporate the coordinates into a detailed final plan. This approach is not a fair comparison to our framework because using the VLLM to extract accurate, actionable coordinates is the more challenging part of this task.

Since the authors did not mention how the coordinates were extracted, and from our previous exploration, using off-the-shelf trained object extraction models such as OWL-ViT did not perform well on VIMABench (Figure ?? shows this fact), we assume that NLaP used information as accurate as human-extracted data. We tried two versions of implementation for this: 1) using gpt-4o to extract this information in the same format, and 2) using ground truth information. For the second approach, we used the ground truth object names from the environment and the ground truth coordinates by mapping the environment state to the pixel coordinate scale. Note that although this approach does not offer a fair comparison to our method, we implemented it to understand how well the planning component performs and to replicate their original results. However, it is important to keep this major difference in mind when interpreting the results.

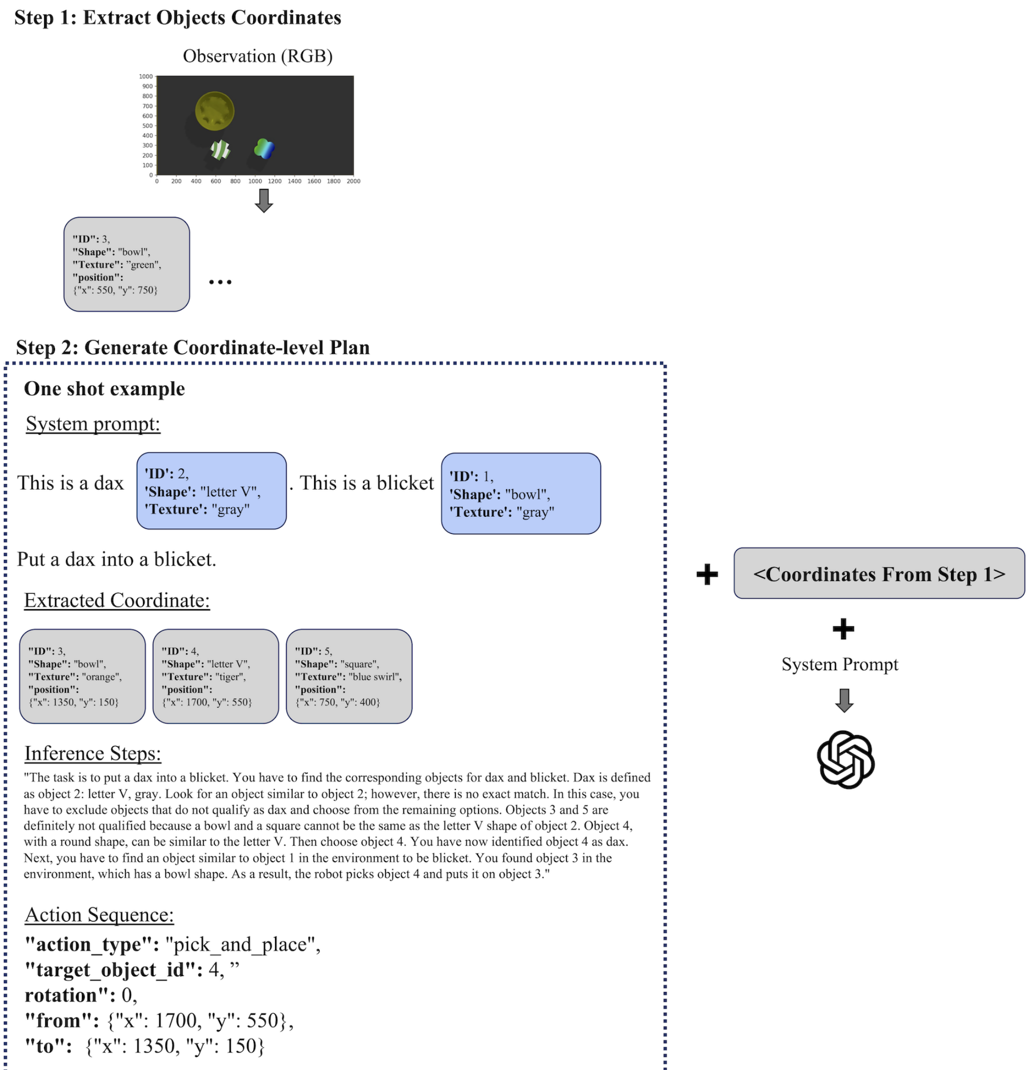


Figure 23: Example - Original Framework of NLaP

Another significant difference between their framework and ours is that the planning component of NLaP does not use any visual information, as shown in Figure 23. In the extraction part, information on objects and their coordinates is derived from visual data, either by human labeling, VLLM, or another model. During the planning phase, the LLM only has access to the textual information. This explains why there wouldn't be a significant difference between using gpt-4o and gpt-3.5-turbo, as gpt-3.5-turbo is already very proficient at planning, and the planning part of the framework would not benefit substantially from switching to gpt-4o.

In our implementation of NLaP using gpt-4o for both coordinate extraction and action sequence generation, however, we added the corresponding visual information of both the extracted information and the one-shot example to facilitate the understanding of VLLM of the environment. The idea of our implementation of this added vision version is shown in Figure 24.

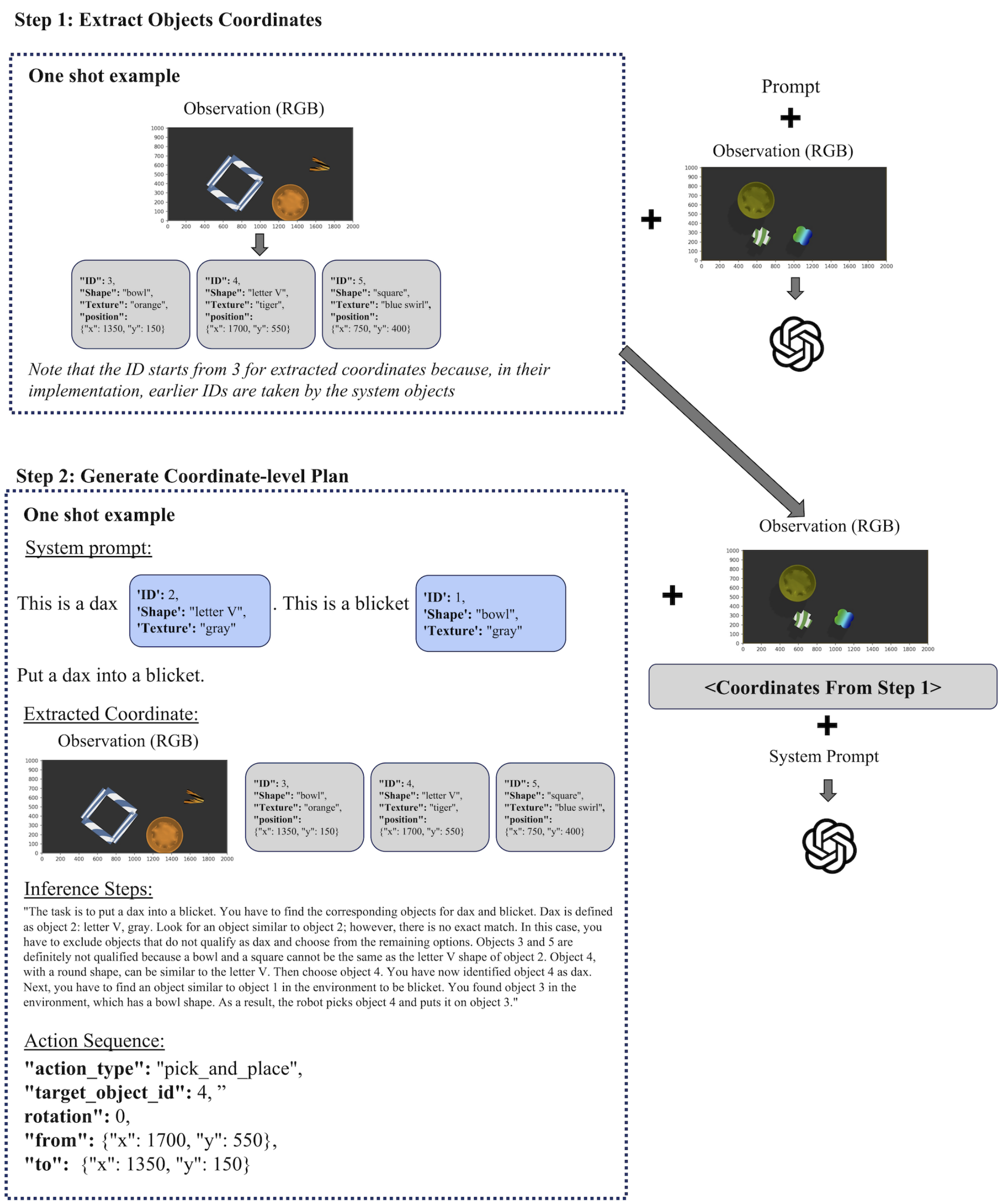


Figure 24: Example - Framework of NLaP with Visual Information Added

Another difference in our experimental evaluation between our method and Natural Language as Policies is that NLaP directly takes the system information of objects for multi-modal prompts. For instance, see an example in Figure 25. In some VIMABench tasks, the prompts can be made multi-modal, and parts of the prompts, usually objects, are not described by words but by images. We used this version of the prompt without any text information for these parts in our evaluation to test the robustness on multi-modal tasks. However, in NLaP, they used the system text information on the shape and texture instead of visual data.

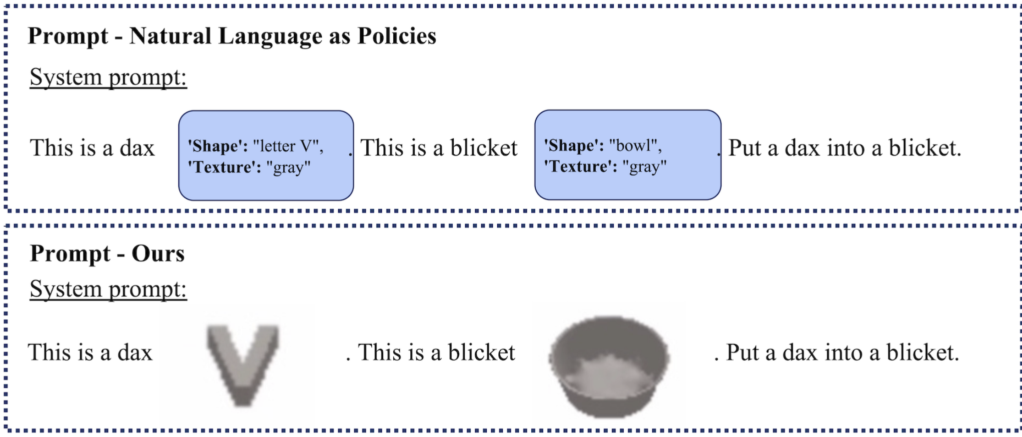


Figure 25: Illustration of the Difference in Multi-modal Prompts: This figure shows the variation in how prompts are constructed between our method and the NLaP system. Our method uses visual information (images) for object description, while NLaP uses system-generated shape and texture information.

One last difference between our methods is that in their prompt, a one-shot example is given. Examples can be viewed in Table V of their paper. The example simply illustrates a typical thought process of a successful execution. They used different examples for different tasks, and during our experiments, we found that sometimes the tasks can be overly similar to the actual task in terms of reasoning, object shape, even object number. For instance, in simpler scenes with two objects, the final desired output is always putting object 3 into object 4 or vice versa. Examples like this may sometimes provide unintended hints that could over-simplify the task.

Table 9: Success Rates Across Different Settings

Task Num	gpt-4o + gpt-4o	gpt-4o + ground truth	gpt-3.5 + ground truth	NLaP Reported	Ours
1: Visual Manipulation	20	100	100	100	100
3: Rotate	30	100	90	93	100
6: Novel Adjective	10	80	60	43	70
7: Novel Noun	40	100	80	80	100
15: Same Shape	0	10	70	80	100
16: Manipulate Old Neighbor	0	60	20	20	90

In Table 9, we present the results of our ablation studies. We used a '+' sign to denote the combination of settings for planning and coordinate extraction, respectively. For example, 'gpt-4o + gpt-4o' represents the setting where we used gpt-4o to extract scene information (as shown by the red box in Figure 22), while 'gpt-4o + ground truth' means that we directly fed the language model with the actual coordinates and system object names.

From the results, we can see that the comparable version of NLaP, where both planning and grounding are done by the VLLM, barely succeeds on VIMABench tasks, even on simple, one-step tasks. It performs significantly worse compared to our method. The failure modes are often caused by both shortcomings in planning and inaccuracies in the position-finding step. In their original implementation, where coordinate-level information is directly gathered from the environment system instead of by a zero-shot VLLM model, switching from gpt-3.5-turbo to gpt-4o achieves slightly better results. This improvement is likely due to gpt-4o's enhanced reasoning capabilities, which are beneficial for more complex tasks, such as identifying multiple old neighbors that require reasoning about relationships.

However, since their implementation primarily relies on textual information extracted from the previous steps rather than vision information during the reasoning phase, the gain from switching to gpt-4o, which excels in vision understanding, is limited. As a result, gpt-4o under the NLaP framework still struggles with tasks involving identifying objects of similar shape. A common failure mode is its insistence that no object has a similar shape.

These results further show that **the multi-agent structure is crucial for our system's overall performance.** Even with perfect system output for localization used by Natural Language as Policies, long-horizon planning with complex reasoning remains challenging without the self-corrective multi-agent structure.

E.2 Comparison with PIVOT

PIVOT (Iterative Visual Prompting Elicits Actionable Knowledge for VLMs) focuses on localization through visual question answering, with minimal emphasis on planning—similar to the role of our grounding team within our hierarchical framework. PIVOT (Nasiriany et al., 2024) introduces an innovative approach to enabling VLMs to localize actionable points or actions by progressively shrinking the action distribution and resampling. The process begins by sampling a set of actions from the action space, which are then mapped onto a 2D image. A VLM is used to select the most promising actions from this set. Based on these selections, a new action distribution is created, and the process is repeated over a fixed number of iterations to refine the actions further.

In their robotic environment implementation, PIVOT handles two versions of localization: one involves finding a multi-dimensional relative Cartesian (x, y, z) coordinate in the action space, and the other involves finding a pixel coordinate in the pixel action space—similar to our approach in VIMABench, where control is based on pixel coordinates rather than relative Cartesian coordinates. For action mapping, PIVOT maps actions to a final endpoint, effectively aligning with the pixel coordinate localization method.

In our comparison, we use VIMABench, where control is based on coordinate-level actions. Therefore, PIVOT's coordinate mapping implementation and the prompts they used on the RAVENS simulator are applied throughout our analysis. There are several similarities and differences between our work and PIVOT that are worth highlighting.

Similarities:

- Both frameworks extract coordinate-level information.
- Both operate in a zero-shot manner without any fine-tuning.
- Both annotate 2D images and provide these annotations to the VLLM to guide its decision-making.

Differences:

- Our framework focuses on both planning and localization, with localization being one component within a hierarchical structure designed to handle long-horizon tasks with complex planning. In contrast, PIVOT **only focuses on localization**, where their prompts typically describe an object or subgoal rather than addressing a broader task.
- PIVOT uses a **single agent** responsible for iteratively selecting a point from a sample of points or action-mapped points. In contrast, our grounding team consists of **multiple agents**, each playing a distinct role in a self-corrective process.
- PIVOT's method can be viewed as a process of shrinking or guiding the sampling distribution closer to the target object, with each iteration's samples based on the previous one (Fig 26). While our method is also iterative, we begin with a point chosen by the grounding manager and refine it iteratively from there (Fig 27), rather than starting with the entire distribution of possible locations.
- PIVOT identifies a **single action point** for the target object, maintaining this as the goal throughout their iterative process. In contrast, our method offers two distinct workflows that the grounding manager can choose from before localization. When selecting an area point, such as a position between a box and a frame, we also employ point selection. However, for object selection, our method first identifies a center point, then determines a **bounding box** of appropriate size, and iteratively refines this bounding box until it is accurate. The grounding manager then selects an actionable point within the bounded area. We found that this bounding box process greatly enhances robustness and precision, especially for smaller objects or manipulation tasks that require more precise control. We further ablate and discuss this in Appendix E.2.

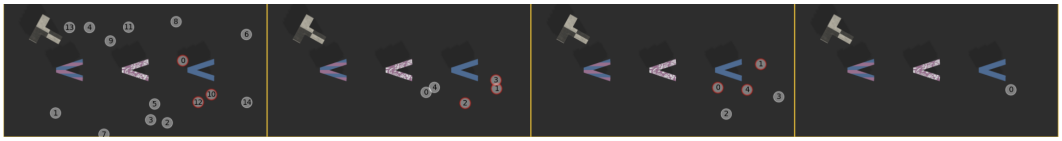


Figure 26: PIVOT Workflow, Blue Letter V

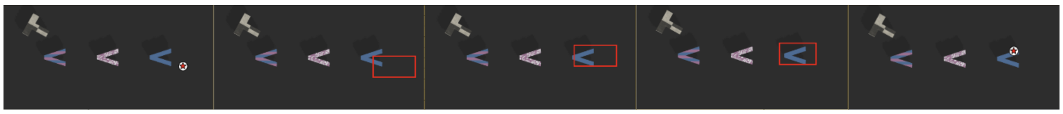


Figure 27: Wonderful Team Workflow, Blue Letter V

Next, we present some quantitative evaluation on object identification results in selected VIMABench environments followed by further discussions on the failure modes.

In Table 10, we compare the experimental results of our method with those from PIVOT. While PIVOT originally utilizes GPT-4V in its framework, we implemented their approach using the more advanced GPT-4O to ensure a fair comparison. Our replication of their framework was carried out to the best of our knowledge to highlight the differences and performance improvements. Additionally, we include results obtained from their official HuggingFace demo to demonstrate the performance of their original implementation. For example output of different grounding approaches, please see 29.

Table 10: Location Grounding Success Rates

Task	PIVOT (gpt-4v) (HF) (%)	PIVOT (gpt-4o) (%)	gpt-4o Direct Output (w/ labeled axes) (%)	Ours (grounding team) (%)
1. Visual Manipulation	10	30	40	90
6. Novel Adj	0	0	20	80
17. Pick in Order then Restore	0	0	10	90

Implementation Details

Uniform Sampling: PIVOT begins by sampling a set of actions from the action space (in VIMABench or RAVENS, as reported in their paper, this involves sampling 2D coordinates), which are then mapped onto a 2D image. A VLM is used to select the most promising actions. Based on these selections, a new action distribution is fitted, and the process is repeated over a fixed number of iterations to refine the actions. Due to the absence of specific details regarding the distribution used in their original implementation, we opted for a uniform sampling strategy. The sampling radius was determined as twice the maximum distance from the average action point to any other point in the set. To ensure alignment with the original method, we also utilized their Hugging Face demo (gpt-4v) to replicate their reported performance.

Parallel Runs: The original study also employs a parallel call strategy. To combine results from different runs, they explored two approaches: (1) fitting a new action distribution from the output actions and returning it, and (2) selecting a single best action using a VLM query. In our implementation, we used the second approach with “3 Iterations 3 Parallel“ combinations to enhance robustness in our comparison. Additionally, while the original implementation uses the same sampling radius for both width and height, we addressed this by defining separate radii for the shorter and longer edges of the input image.

Grounding Team Only: Since PIVOT's framework is primarily comparable to our grounding team, which focuses on processing object descriptions rather than broader tasks, we isolated the grounding component for a direct comparison with their method.

Success Evaluation: For evaluation, we conducted 10 runs on different objects from a set of varied initial frames. A task was considered successful if the center point label of each target object had at least half of its area within the object's boundary or if the center point fell within a specific range around the target area center, ensuring successful picking.

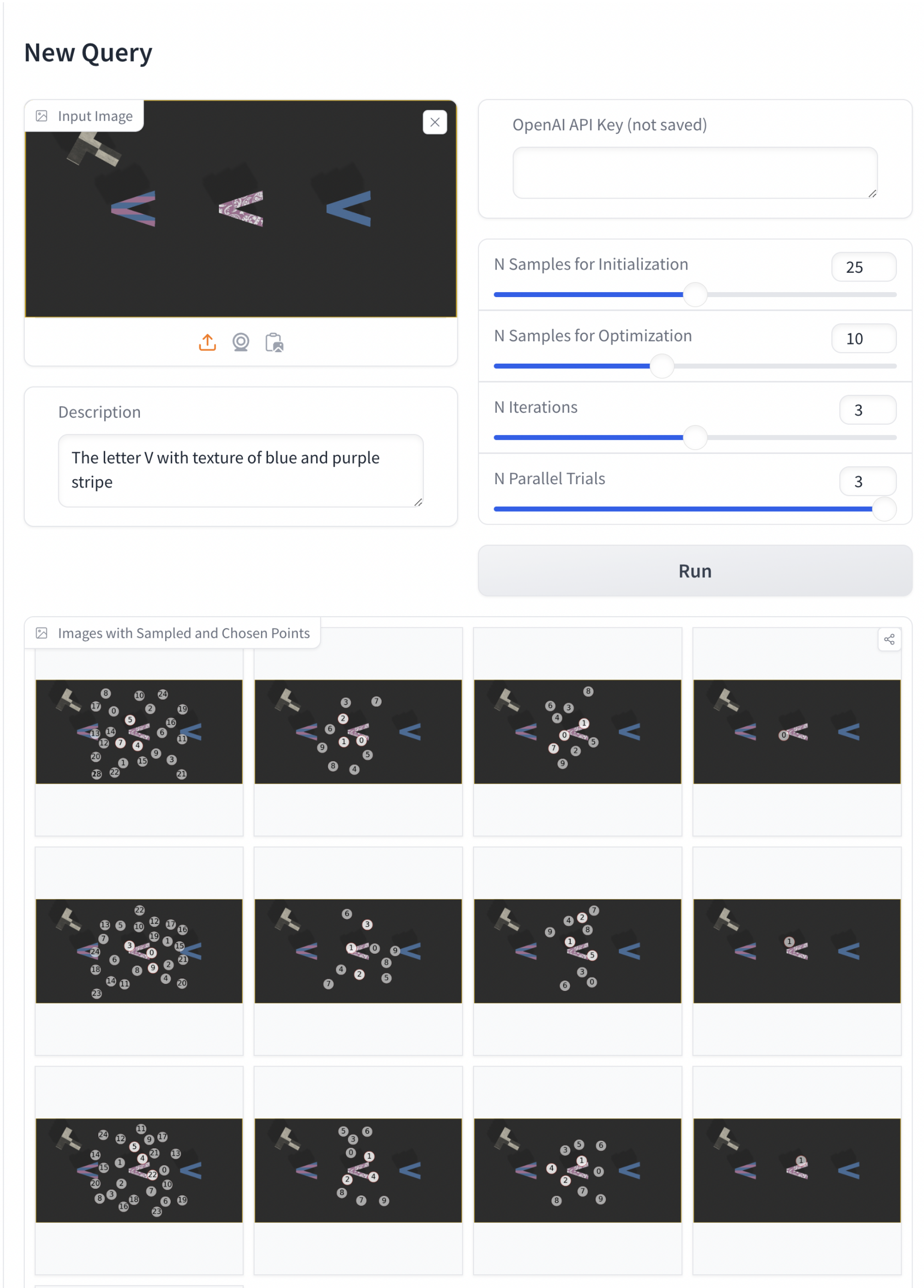


Figure 28: Screenshot of HuggingFace PIVOT Demo

Failure Mode Discussions

It's notable that PIVOT's output on tabletop tasks does not over-perform the direct output from GPT-4o. However, this is with the help of the labeled coordinate system, which significantly enhances precision in quantification, as discussed in our motivation section. We further discuss the possible explanations of PIVOT failures:

Incomplete Sampling Coverage: In 28, when attempting to select the left object, the initial sampling failed to provide sufficient coverage, with the majority of points being sampled from the center of the image and scattering on the purple paisley letter "V" instead of the target object with blue and purple stripes. As a result, subsequent iterations were confined to a suboptimal region, ultimately leading to poor final results.

Difficulty in Recovery: During our implementation, we identified a critical limitation in the sampling strategy: if the sampling radius is too small, it becomes difficult to recover from an inadequate initial selection. Conversely, if the sampling radius is too large, the framework struggles to converge, as the sampled actions may scatter too broadly, reducing the effectiveness of the refinement process.

Lack of Iterative Continuity: Another factor that may explain PIVOT's low performance in precise location finding is the lack of continuity between iterations. Although the new set of actions is sampled from a distribution fitted using previously selected promising actions, there is a notable discontinuity in the process. For instance, if a good point is identified during one iteration, it is not guaranteed to be preserved in subsequent iterations. The framework's fixed number of resampling processes means it cannot exit the process once a good point is found, potentially resulting in the loss of successful actions. This resampling process can lead to promising actions being either diluted or completely discarded in the next round due to inherent randomness, causing inefficiencies and inconsistencies as the framework may fail to build on previous successes.

Messy Annotations: Additionally, the framework's annotations can become cluttered, leading to a loss of crucial information from the original image. Unlike our approach, which maintains a clear connection to the original image to preserve full context, PIVOT's method can lose track of the overall scene, making it difficult to refine action points effectively. This loss of context can be particularly detrimental in scenarios where precision and consistency are critical.

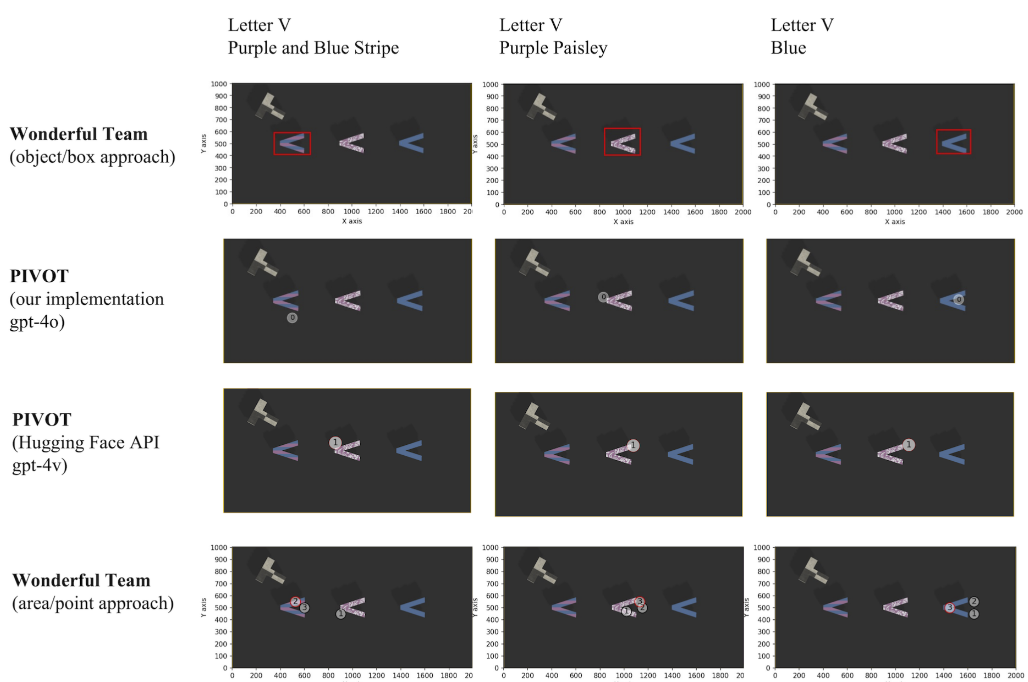


Figure 29: Example Outputs - Wonderful Team vs PIVOT

Point Selection vs. Bounding Box: Since the PIVOT method is inherently more similar to our area/point approach discussed earlier—where points are selected throughout the process without the aid of bounding boxes—we further compare PIVOT's outputs with both our bounding box approach and our point approach. Figure 29 provides insight into how these methods perform relative to each other. While both PIVOT and our area/point approach can get reasonably close to the desired objects, they often lack the precision required for tasks involving small objects or when execution demands more accuracy than just proximity to the object.

In Figure 30, we present example executions using the results from these methods. The task involves stacking the purple and blue striped letter “V” on top of the blue letter “V,“ followed by stacking the purple paisley letter “V” on top. For this execution, we used the PIVOT results from our implementation using gpt-4o, as the HuggingFace outputs were less reliable, with all points concentrated on the same object. The execution screenshots reveal that points not accurately placed on the object lead to failures in picking it up. On the bottom row of Figure 30, even though both points for the first pick-and-place action are technically correct, the misalignment causes the stacking task to partially fail, as the letters “V“ are not properly aligned, resulting in an unsuccessful stack.

These results highlight the importance of considering whether a bounding box is needed in the iterative process. With the current level of visual reasoning skills in models, we found that incorporating a bounding box significantly enhances precision, reduces hallucinations, and adds robustness to the execution.

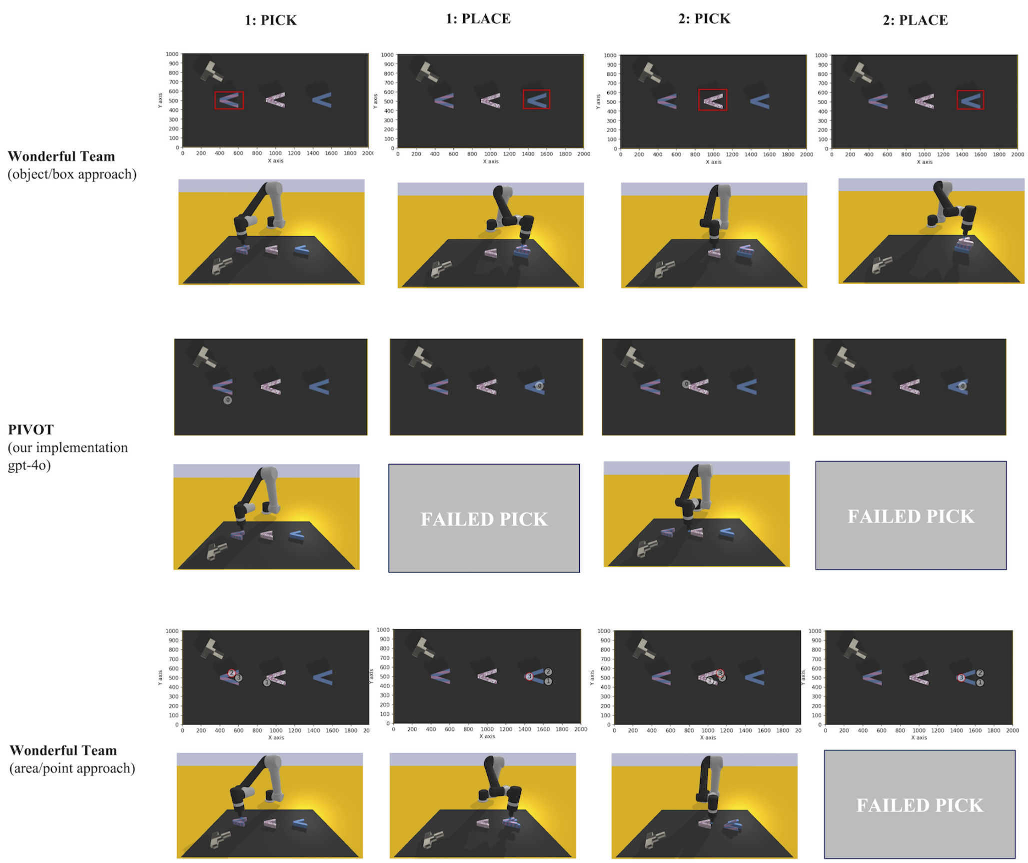


Figure 30: Example Executions - Wonderful Team vs PIVOT

These limitations underscore the shortcomings of the PIVOT framework and highlight the necessity of a more guided and context-aware approach, as implemented in our method.

E.3 Comparison with Language Models as Zero-Shot Trajectory Generators

E.3.1 Key Differences

In Language Models as Zero-Shot Trajectory Generators (Kwon et al., 2024), the task is given to a LLM (gpt-4) in text form. After this, the LLM identifies task-related objects and call an object detection API to retrieve the information about these objects (xyz, height, orientation etc). Using this retrieved information, the LLM starts to plan. In particular, it achieves planning by writing python scripts to generate a trajectory to be executed.

When compared to Wonderful Team, there are a few key differences.

First, the authors employed gpt-4, which does not have vision capability. This means when LLM is making decisions on what objects to detect and generating plans, it does not have any context of the environment except for the one-line command from the user. To improve on the lack of context when making plans, the authors could swap gpt-4 with gpt-4o and provide an image of the environment. This way, the VLLM could identify any task-related objects that are NOT in the command for object detection.

However, even in this case, there are still some issues with the detection process. We experimented with swapping our grounding team with detection models, such as OWL-ViT or langSAM, in the early stage of our research. These methods fail to detect almost all objects that cannot be directly described within a few words. As a concrete example of the problems we encountered with this approach, imagine a user issuing the command: “Pick up the thing to the left of the bottle.” Upon reading this command, the detection module will try to find “the thing” and fail, because obviously such an abstract concept can not be encoded into a detection module.

Language Models as Zero-Shot Trajectory Generators uses a single-agent system, where one agent is responsible for generating plans based on user commands. While this method can work under certain conditions, it has inherent limitations, particularly in handling complex, ambiguous instructions and managing long-horizon tasks, especially those that require detailed contextual understanding. In contrast, our system employs a multi-agent architecture, where different agents specialize in specific tasks such as localization, planning, and validation.

E.3.2 Single Agent vs Multi-Agent

When comparing the single-agent approach, as exemplified by models like Language Models as Zero-Shot Trajectory Generators, to our multi-agent system, it’s important to recognize the distinct challenges each method addresses. Single-agent systems typically solve a more straightforward problem that focuses solely on planning. These systems rely on a separate detection module to identify objects, followed by planning over these detections. While this approach can work in controlled settings, it often leads to instability and misinterpretation of language instructions, particularly when the model encounters more complex or ambiguous commands.

In contrast, our multi-agent system integrates both planning and localization directly within the framework, using Vision-Language Models (VLLMs) to extract object location information. This direct extraction requires a multi-agent setup, where each agent is responsible for a specific aspect of the task, incorporating additional confirmation steps and sub-loops to ensure accuracy. This multi-agent architecture not only addresses the grounding problem but also significantly enhances the system’s capability to solve complex, long-horizon tasks, as demonstrated in our evaluations. For instance, in the “manipulate old neighbor“ task from VIMABench, even when given ground truth coordinates, a single-agent system using GPT-4o within the NLaP framework often failed to generate successful plans (see Table 9).

E.4 Benefits of Using a Multi-Agent System

The multi-agent system we propose offers several key advantages over single-agent systems:

1. Suitability for Robotics Tasks. A multi-agent system is particularly well-suited for robotics tasks because these tasks typically involve distinct and varied challenges that require different approaches. Unlike language-only tasks, which may be more uniform, robotics tasks often demand specialized strategies for different components, such as object detection, manipulation, and planning. By employing a multi-agent system, each aspect of the task can be handled by an agent specialized in that area, improving both the efficiency and accuracy of the system. Moreover, the ability of agents to communicate and validate each other's work leads to more reliable decision-making and reduces the likelihood of errors, especially in complex, dynamic environments.

2. Simplified System Complexity. At first glance, a multi-agent system might seem more complex than a single-agent approach. However, by dividing the task into smaller, more manageable components, each agent can focus on a specific, well-defined role, which actually simplifies the overall system. This division of labor is especially beneficial in robotics, where different aspects of a task require different strategies. By tailoring each agent's prompts and tasks to their specific role, we avoid the pitfalls of trying to handle everything within a single, monolithic prompt. For instance, when a single agent is responsible for object detection, manipulation, and planning, it often struggles with precise location identification and may produce partially incorrect or infeasible plans.

3. Effective Communication and Validation. Communication between agents is another significant advantage of our multi-agent approach. Instead of an agent re-evaluating its own output — potentially leading to unnecessary adjustments or confusion — different agents can validate the outputs independently. This reduces the risk of hallucinations, which can occur when an agent is overly influenced by its previous decisions. For example, when a verification agent (or box checker) evaluates the outputs from the supervisor (or box mover), it treats these outputs as a new query, asking questions like "Is A better than B?" or "Is this action feasible?" This approach contrasts with single-agent systems, where the agent might simply consider whether to fix an existing plan, a situation that often leads to further errors.

4. Enhanced Self-Correction. One of the primary strengths of a multi-agent system is its ability to self-correct through agent interaction. In a single-agent system, the same agent must generate a plan and then evaluate it, which can lead to confusion and unnecessary revisions due to hallucinations or biases from previous outputs. In contrast, our multi-agent system allows agents to communicate and validate each other's outputs, significantly reducing the likelihood of such errors. For example, if a VLLM proposes an incorrect object location, this often results in a failed trajectory in 78% of cases. However, when a team of agents iteratively improves the target locations, the success rate increases to 93% (see page 35, Table 4).

5. Improved Memory Management. In a multi-agent system, no single agent is burdened with managing the entire context or retaining all information, which can lead to hallucinations or errors. For example, in the "pick in order then restore" task, the success rate was only 40% without a memory module, but it increased to 90% when a dedicated memory agent was included. This demonstrates how distributing responsibilities among agents enhances both performance and reliability by reducing the cognitive load on any single agent.

E.4.1 EXPERIMENTAL COMPARISON IN FETCH

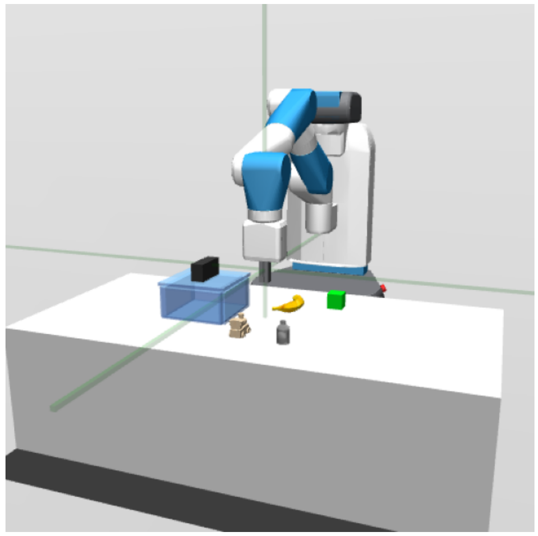


Figure 31: Default View of Fetch Environment with a Box with a Lid

We further compared our methods in a Gymnasium environment involving a box covered by a lid.

Environment: The robot used is a 7-DoF Fetch Mobile Manipulator equipped with a two-fingered parallel gripper. The setup includes a closed box with a lid and four other objects placed on the table. See Figure 31 for an example setup.

Task: The task is to place one or two of the objects into the box.

Example Prompt: "Place the wooden toy train and the rightmost object inside the small blue box with a lid and a black handle." (The exact prompt depends on the target objects.)

Why This Task is Challenging:

- It requires accurate 3D estimation. Although this can be partially addressed by using a 2D image with a depth array, there can be challenges when converting 3D information to 2D. Even small deviations in this process can lead to significant errors in execution.

- Items are positioned at different height levels, so collision avoidance must be carefully considered. This is particularly important because the box is quite deep, requiring a thoughtful approach to placing objects inside.

- Correctly identifying the components of the environment, including the box lid, is difficult. The black handle on the lid is very small and requires precise detection for successful execution. Additionally, the handle's common shape and color may cause it to be misidentified or overlooked.

- The plan needs to include the step of removing the lid, which is often omitted. Moreover, the plan should identify an empty area on the table to place the lid without displacing other objects.

Planning Results:

In the example task, where the goal is to place the wooden toy train and the rightmost object inside the box, the plan generated by Wonderful Team using the prompt, after validation with the verification agent, is shown in Figure 32(b). For comparison, the plan generated with the exact same task prompt by our system is shown in Figure 32(a). We will further discuss the results in the last section.

Step-by-Step Trajectory Plan

Step 1: Moving to the Wooden Toy Train

- Move to a position directly above the wooden toy train.
- Lower the gripper to a suitable height.
- Close the gripper to pick up the train.
- Raise the gripper to avoid collisions.

Step 2: Moving to the Rightmost Object

- Move above the rightmost object.
- Lower the gripper to a suitable height.
- Close the gripper to pick up the object.
- Raise the gripper to avoid collisions.

Step 3: Moving to the Small Blue Box

- Move above the small blue box.
- Lower the gripper to just above the inside of the box.
- Open the gripper to release the first object.
- Raise the gripper slightly and repeat for the second object.
- Open the gripper to release the second object.

(a) Plan Generated by Trajectory Generator

```
output["verified_high_level_plan"]
```

```
['Pick up the box lid',
 'Place the lid at an empty spot to the side',
 'Pick up the wooden toy train',
 'Place the toy train inside the box',
 'Pick up the green cube on the right',
 'Place the green cube into the box',
 'Pick the box lid from the side',
 'Place the lid back onto the box']
```

(b) Plan Generated by Wonderful Team

Figure 32: Comparison of Plans Generated by Trajectory Generator and Wonderful Team

Detection Results:

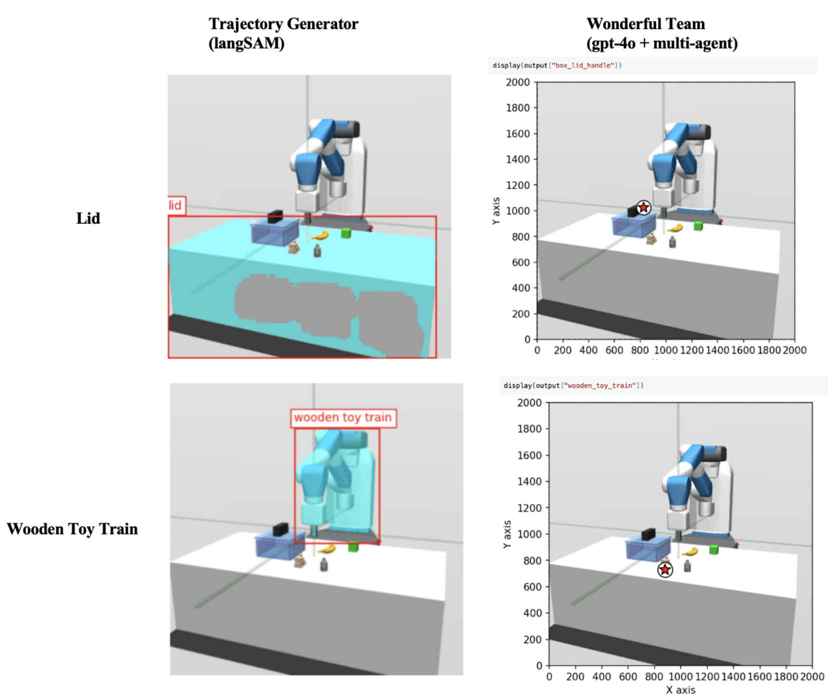


Figure 33: Examples of Object Detection. Check Google Colab notebooks for more example results for Wonderful Team and Trajectory Generator.

Success Rate Results:

Table 11: Success Rates on Fetch Box

Method	Success Rate (%)
Wonderful Team (single attempt)	50
Wonderful Team (re-planning allowed)	80
Trajectory Generator (single attempt)	0
Trajectory Generator (re-planning allowed)	5

Summary of Findings:

- **Trajectory Generator (Planner):** The planner often fails to understand the implied requirements in the task instruction and is only capable of considering the explicit commands. See Figure 32(a) for an example. Without the command to remove the lid, the planner starts by picking up a target object instead of opening the box to prepare for later steps. In addition to this, the planner also assumes that the gripper can hold two objects at a time before placing them down in the specified container, which is a result of not having access to the environment in context.
- **Trajectory Generator (LangSAM):** This model struggles to correctly identify many objects. See Figure 33 for instance, when asked to find the wooden toy train, it points to the Fetch robot; when asked to locate the lid, it points to the entire table. Similarly, when asked to identify the rightmost object, it again points to the Fetch robot, and when asked to locate the tomato soup can, it points to the mustard bottle.
- **Wonderful Team's Performance:** Wonderful Team achieves a 50% success rate on this task. The main failure mode arises from the difficulty in integrating the depth camera for accurate position estimation, which sometimes results in missed targets.
- **Impact of Replanning Module:** When we introduced a replanning module, Wonderful Team's success rate improved to 80%.