MALMM: Multi-Agent Large Language Models for Zero-Shot Robotic Manipulation

Harsh Singh*†[1], Rocktim Jyoti Das†[1], Mingfei Han[1], Preslav Nakov[1], Ivan Laptev[1]

***Abstract*— Large Language Models (LLMs) have demonstrated remarkable planning abilities across various domains, including robotic manipulation and navigation. While recent work in robotics deploys LLMs for high-level and low-level planning, existing methods often face challenges with failure recovery and suffer from hallucinations in long-horizon tasks. To address these limitations, we propose a novel multi-agent LLM framework, *M*ulti-*A*gent Large *L*anguage *M*odel for *M*anipulation (MALMM). Notably, MALMM distributes planning across three specialized LLM agents, namely high-level planning agent, low-level control agent, and a supervisor agent. Moreover, by incorporating environmental observations after each step, our framework effectively handles intermediate failures and enables adaptive re-planning. Unlike existing methods, MALMM does not rely on pre-trained skill policies or in-context learning examples and generalizes to unseen tasks. In our experiments, MALMM demonstrates excellent performance in solving previously unseen long-horizon manipulation tasks, and outperforms existing zero-shot LLM-based methods in RLBench by a large margin. Experiments with the Franka robot arm further validate our approach in real-world settings.**

I. INTRODUCTION

Robotic manipulation has seen impressive advancements, enabling agents to handle increasingly complex tasks with greater precision and efficacy. Current solutions, however, often struggle with generalization, in particular when using imitation learning for policy training [1], [2]. Such methods typically excel at specific tasks but lack the adaptability to handle new tasks. One major drawback of imitation learning is the labor-intensive and time-consuming process for data collection, which limits the scalability of resulting policies. Moreover, training task-specific manipulation policies typically require thousands of training episodes [1], [2], making the approach computationally expensive and inefficient. To cope with the generalization, robotics policies should demonstrate a deeper understanding of their environment. This involves recognizing and grounding relevant objects and understanding the relationships between them [3]. Equipped with this knowledge, policies can then plan and execute actions more efficiently while adapting to changes in the environment and new task requirements.

Recent advancements in Large Language Models (LLMs) have demonstrated remarkable generalization and reasoning capabilities across diverse domains such as commonsense, mathematical, and symbolic reasoning [4]. These models, particularly when scaled to billions of parameters, exhibit emergent abilities to break down complex tasks into simpler steps through structured reasoning techniques such as *chain of thought* [5]. LLMs have already shown promise in high-level task planning in different domains, suggesting their potential for flexible and versatile robotic manipulation. Yet, recent LLM-based methods for robotic planning face multiple challenges. One important issue is the tendency of LLMs to produce incorrect high-level plans and low-level control. Additionally, LLMs suffer from hallucinations in long-context generation [6], which is often observed in closed-loop LLM systems. As a result, they may disregard geometric constraints and the parameters of predefined functions or may even lose the sight of the goal.

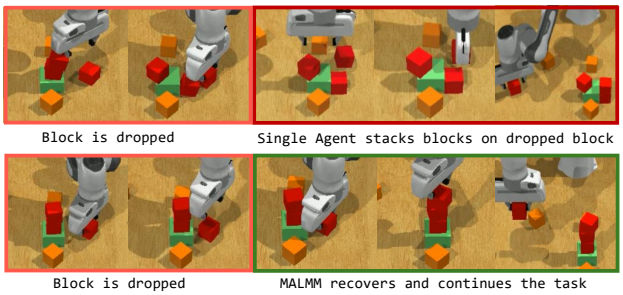


Fig. 1: Examples of executing *"Stack four blocks at the green target area"* task by the Single Agent LLM (top) and our Multi-Agent MALMM framework (bottom). MALMM recovers after dropping one block and continues stacking above the target area, while the Single Agent mistakenly continues stacking blocks on top of the dropped block.

In this work, we propose a **M**ulti-**A**gent Large **L**anguage **M**odel for **M**anipulation (MALMM) that leverage the collective intelligence and the specialized skills of multiple agents for complex manipulation tasks. Our framework incorporates agents dedicated to high-level planning, low-level code generation and a supervisor that oversees transitions between other agents and tools. We show that through the use of multiple specialized agents in a multi-agent setup, we are able to mitigate the hallucination issues observed in the case of a single agent, as shown in Fig. 1.

Our contributions can be summarized as follows:

- We introduce the first multi-agent LLM framework for robotic manipulation MALMM, equipped with specialized agents that bring collaborative and role-specific capabilities to complex manipulation tasks.
- We demonstrate the advantages of the proposed multi-agent framework through systematic ablation studies on

*Corresponding author.
†Equal contribution.
[1]Mohamed bin Zayed University of Artificial Intelligence, UAE
email: firstname.lastname@mbzuai.ac.ae,
Project Page: https://malmm1.github.io/

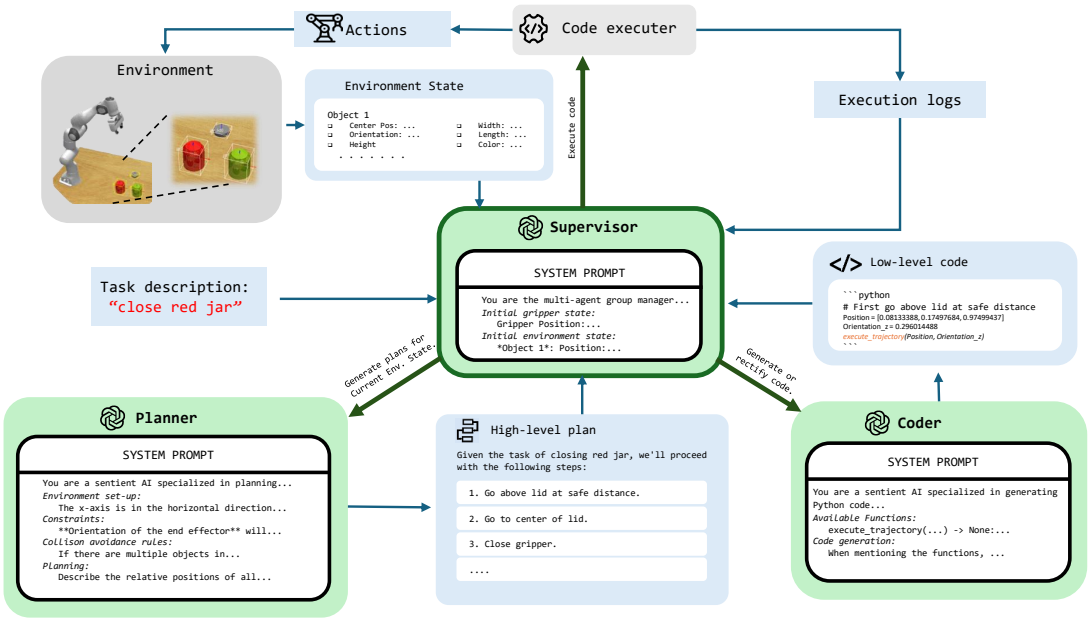


Fig. 2: An overview of our multi-agent system, **MALMM**, which consists of three LLM agents—**Planner**, **Coder**, and **Supervisor**—and a **Code executor** tool. Each agent operates with a specific system prompt defining its role: (1) the **Planner** generates high-level plans and replans in case of intermediate failures, (2) the **Coder** converts these plans into low-level executable code, and (3) the **Supervisor** coordinates the system by managing the transitions between the Planner, the Coder, and the Code executor.

tasks with varying horizons and complexity.
- We evaluate MALMM in challenging zero-shot settings both in a simulation and in the real world, and we show substantial improvements over the state of the art.

II. RELATED WORK

A. Language Grounded Robotics

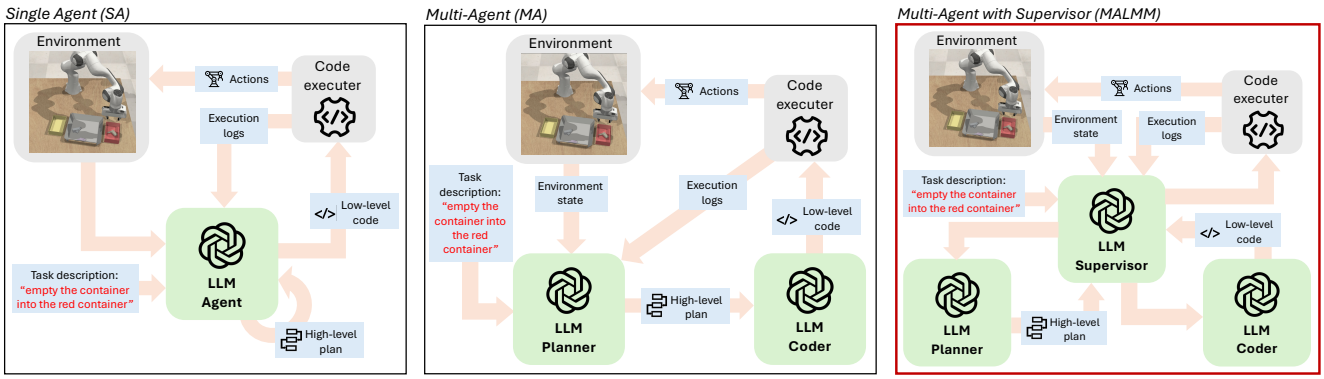
Language instructions enable the definition of complex robotics tasks with compositional goals [7] and support scalable generalization to new tasks [8]. The literature around language grounding is vast, ranging from classical tools such as lexical analysis, formal logic, and graphical models to interpreting language instructions [7], [9]. Recently, much effort has focused on adopting the impressive capabilities of Large Language Models (LLMs) to language grounding in robotics [8]. Additionally, recent advancements have benefited from pre-trained LLMs thanks to their open-world knowledge, tackling more challenging tasks such as 3D robotic manipulation and leveraging code generation capabilities to produce high-level, semantically-rich procedures for robot control.

B. LLM for Robotics

Language Models have been used for various robotics purposes including the definition of reward functions [10], task planning [11], [12], failure summarization and guiding language-based planners [13], and policy program synthesis [14]. VoxPoser [15] and Language to Rewards [16] used LLMs for generating reward regions for assisting external trajectory optimizers in computing trajectory. Our work is most related to methods using LLMs for manipulation planning. Most of such work [12], [15] relies on pre-trained skills, motion primitives, and trajectory optimizers and has focused primarily on high-level planning. The closest to our approach is *Language Models as Zero-Shot Trajectory Generators* [11], which deployed a language model to generate high-level plans and then convert these plans into low-level control. However, LLMs suffer from hallucinations, which affect long-horizon task planning. Moreover, [11] assumed the correct execution of each step and did not account for occasional failures or unforeseen changes in the environment. Our evaluation shows sizable improvements of MALMM over [11] thanks to its multiple specialized agents and intermediate environment feedback.

C. LLM-Based Multi-Agents in Robotics

Recently, several studies have focused on using LLMs for sequential decision-making and complex reasoning tasks [17]. There is also an emerging field of using multiple agents driven by LLMs to solve complex tasks, including robotic manipulation, in a collaborative manner [18]. In most of the work in robotics, LLMs have been used in multi-robot collaboration and communication. Moreover, Reinforcement Learning (RL) policy agents are also used in collaboration with LLMs, which limits their generalization to new tasks and environments [19]. In contrast, we propose a multi-agent framework, MALMM, which incorporates three role-specific LLM agents and enables zero-shot execution of previously unseen robotic manipulation tasks.



(a) (b) (c)

Fig. 3: Agents for robotic manipulation: The figure illustrates three LLM-based manipulation frameworks: **Single-Agent (SA)**, **Multi-Agent (MA)**, and **Multi-Agent with Supervisor (MALMM)**, with the different number of agents in each framework. All three frameworks begin by receiving an input command and the initial environment observation. Each framework iteratively generates a high-level plan along with corresponding low-level code. After each intermediate step, the frameworks use updated environmental observation to detect failures and replan as needed until the task is completed.

III. METHOD

Large Language Models (LLMs) have recently emerged as a universal tool for a variety of tasks, including robotic navigation and manipulation [20], [15], [14], [11]. Such models exhibit surprising generalization abilities and support zero-shot solutions for new tasks. While several recent methods have explored LLMs for high-level task planning [21], [12], other methods attempted to bring the power of LLMs to the low-level trajectory planning [14], [11]. We follow these works and leverage LLMs to precisely control the end effector to solve complex and previously unseen manipulation tasks.

Trajectory planning is a complex problem that requires reasoning about the precise shape, position, and semantics of the objects as well as understanding object interactions. To solve this problem, LLM agents can be instructed to handle information about objects and robots by *text prompts*. To harness the complexity of the problem, here we propose to address the manipulation problem with multiple LLM agents. Inspired by recent work on multi-agent LLMs, we design agents specialized in different aspects of the manipulation problem and connect them into a framework. Below, we describe our proposed **M**ulti-**A**gent Large **L**anguage **M**odel for **M**anipulation (MALMM) framework in detail (see Fig. 2).

A. Preliminaries

Our goal is to design a LLM framework capable of solving previously unseen manipulation tasks defined by natural language instructions. We assume access to a robotic environment equipped with a Franka robot arm supporting manipulation actions of closing the gripper, opening the gripper, and changing the gripper's position and orientation. Observations of the environment are given either as 3D poses of objects and grippers obtained from a simulator or as a 3D point cloud obtained from an RGBD camera. In the latter case, we estimate the object poses and potential grasping poses from the point cloud. We also assume access to a LLM that can process language instructions together with environment observations in the form of text.

B. MALMM: Multi-Agent Manipulation

The core motivation of our work is to investigate how a multi-agent framework can leverage the collaborative and role-specific reasoning capabilities of multiple LLM agents to complete complex manipulation tasks. Below, we first introduce a single-agent architecture and then we propose its multi-agent extensions. The single- and multi-agent architectures considered in our work are illustrated in Fig. 3.

Single Agent (SA). Single Agent is adopted from [11] by environment-specific (RLBench [22] or real-world) prompt tuning. Prompting LLMs to interpret natural language instructions in the form of executable primitive actions has been shown successful for a variety of tasks, including image understanding [23] and numerical reasoning [24]. Similarly, LLMs can be prompted to interpret embodied tasks, e.g., *open a wine bottle* and convert them into a sequence of primitive manipulation actions. A version of such a system with a single LLM agent is outlined in Fig. 3(a). Here, an LLM is first prompted to break down the language instruction into a sequence of steps. It then uses its code generation capabilities to translate each step into executable Python code, using predefined functions to control the end-effector. This code is then sent to a Python interpreter that executes the steps in the environment. After each step, the LLM receives new observations from the environment and proceeds in a loop with planning and code generation until meeting termination criteria.

Multi-Agent (MA). A Single Agent performing multiple roles struggles to excel in all of them. To address this issue, we propose two specialized LLM agents with shorter role-specific contexts: the *Planner* and the *Coder*, see Fig. 3(b).

The *Planner* breaks down the language instructions into a sequence of manipulation steps while the *Coder* iteratively translates these steps into an executable Python code. After each intermediate step, the *Planner* detects potential failures and re-plans according to new observations of the environment.

Multi-Agent with a Supervisor (MALMM). Our final multi-agent architecture (MALMM) extends MA with a *Supervisor* agent that coordinates the *Planner*, the *Coder*, and the *Code Executor*, as shown in Fig. 3(c). The *Supervisor* decides which agent or tool to activate next based on the input instructions, the roles of the individual agents, the environmental observations, and the entire chat history of the active agents.

C. Multi-Agent Prompting

Each agent is provided with a task-agnostic system prompt. The agents rely solely on their internal world knowledge for reasoning and decision-making. For prompt construction, we draw inspiration from [11] and its study of LLM-based trajectory generation for robotic manipulation. We adapt the prompt according to our environments (RLBench and real-world). Note that unlike other recent work [14], [15], we do not provide the agents with any examples for in-context learning and apply MALMM to new tasks without any changes, i.e., in a zero-shot mode.

Each agent's prompt is specifically designed to suit its role. Since LLMs require step-by-step reasoning to solve tasks, the *Planner* is prompted to generate steps that define the intermediate goals needed to complete the task. MALMM perceives the 3D environment using object poses provided either by the internal state of the simulator or by analyzing the visual observations (see Sec. III-D). Therefore, the *Planner* is given a detailed description of the environment's coordinate system, enabling it to interpret the directions from the gripper's perspective. Given the limited exposure of LLMs to grounded physical interaction data in their training, LLM agents often fail to account for potential collisions. To address this, we include generic collision-avoidance rules in the *Planner* prompts. Moreover, in order to handle intermediate failures, primarily due to collisions or missed grasping, we prompt the *Planner* to evaluate the outcomes after each intermediate step and, if necessary, to replan based on the updated observations.

Finally, the *Supervisor* agent is prompted to manage the framework, coordinating the transitions between the LLM agents, the *Planner*, the *Coder*, and the *Code Executor* to ensure successful task completion.

D. Environmental Observations

LLMs trained on textual inputs cannot directly perceive or interpret 3D environments. Our agents receive information about the environment either from the internal state space of a simulator or from visual observations.

State-space observations. In this setup, the LLM agents have direct access to the simulator's state information. The observations are provided as 3D bounding boxes, along with the colors of the scene objects, and include the position, the orientation, and the open/closed state of the gripper.

Visual observations. To apply MALMM in real-world settings, we restrict observations to the front camera images and the 3D point clouds obtained from an RGB-D sensor, and then use pre-trained foundation models to extract information about scene objects. To this end, we use *gpt-4-turbo* [25] to obtain an object list relevant to the instruction text, and then deploy LangSAM [26] to generate image segmentation masks for the objects, e.g. *block* or *red jar* in the task instruction. We then segment the 3D object point clouds by projecting the 2D segmentation masks into the 3D space. To compute accurate object-centric grasping poses, we apply the M2T2 [27] model and predict grasps given 3D point clouds of target objects. We use the obtained gripper poses to control the gripper during grasping. To facilitate object placement, we estimate the 3D bounding box of the target object in the environment based on the task instructions. For example, in the task *close the red jar*, the target object would be the *red jar*. We extract the 3D bounding box directly from the object's point cloud, and use it to guide the placement process. We leverage these visual observations while conducting experiments in both simulated and real-world settings.

IV. EXPERIMENTS

We evaluate the accuracy of MALMM in a zero-shot settings, i.e. when solving diverse set of previously unseen tasks defined by a short text description.

A. Implementation Details.

We use *gpt-4-turbo*[1] [25] to drive the LLM agents in all our experiments. Additionally, we report the results of MALMM using LLaMA-3.3-70B [28] to demonstrate the performance of our framework with an open-source LLM model. For developing the multi-agent framework, we used AutoGen [29], which is an open-source programming library for building AI agents and facilitates collaboration between multiple agents to solve complex tasks. To perform zero-shot evaluation, we do not fine-tune our agents, and we use no training data for in-context learning. We initially developed our prompts for the *Stack Blocks* task and used them for other tasks without any task-specific tuning. MALMM generates 3D waypoints, while the trajectories are computed and executed using a motion planner, following the approach commonly used in RLBench. Our code, prompts, and additional results are available from the project webpage [30].

B. Environment and Tasks.

We conduct the simulation in CoppeliaSim, interfaced via PyRep, using Franka Panda robot with a parallel gripper. The setup incorporates RLBench [22], a robot learning benchmark that provides various language-conditioned tasks with specified success criteria. For evaluation, we sampled 9 RLBench tasks, which feature diverse object shapes and

[1]The experiments were conducted in June 2024 using the latest checkpoint.

TABLE I: Success rate for zero-shot evaluation on RLBench [22]: The table highlights the best-performing method for each task in **bold** and the second-best-performing method is underlined. **Symbols:** † denotes LLaMA-3.3-70B as a base model, ‡ denotes GPT-4-Turbo as a base model.

Methods	Basketball in Hoop	Close Jar	Empty Container	Insert in Peg	Meat off Grill	Open Bottle	Put Block	Rubbish in Bin	Stack Blocks	Avg
CAP [14]	0.00	0.00	0.00	0.08	0.00	0.00	0.76	0.00	0.00	0.09
VoxPoser [15]	0.20	0.00	0.00	0.00	0.00	0.00	0.36	0.64	0.32	0.17
Single Agent (SA) [11]	0.52	0.40	0.36	0.24	0.44	0.80	0.92	0.48	0.20	0.50
MALMM (†)	0.84	**0.88**	0.60	**0.80**	0.64	0.84	0.84	0.56	0.32	0.70
MALMM (‡)	**0.88**	0.84	**0.64**	0.68	**0.92**	**0.96**	**1.00**	**0.80**	**0.56**	**0.81**

Basketball in Hoop | *Close Jar* | *Empty Container* | *Insert onto Square Peg* | *Meat off Grill* | *Open Wine Bottle* | *Put Block* | *Put Rubbish in Bin* | *Stack Blocks*

Fig. 4: Illustration of the **nine** RLBench [22] tasks used in our evaluation, featuring diverse tasks with varying task horizons and different object shapes.

task horizons. Fig. 4 shows snapshots of the nine considered tasks.

In our state-space observation setup, the input observations are captured as bounding box coordinates and object orientations relative to the simulator's ground truth reference axes. To obtain visual observations, we used a single camera with a frontal view of the scene, as illustrated in Figure 4. The point cloud is derived from the depth map corresponding to this view.

In the real-world setup, we evaluated five tasks on a tabletop using a 7-DOF Franka Emika Panda Research 3 robot equipped with a parallel jaw gripper. Three of these tasks are identical to those that use vision-based observations from the simulator, while two are new. We use an Intel RealSense D435i RGB-D camera to capture the frontal view and the panda-py [31] library to control the robot arm.

C. Baselines.

We compare our approach to three state-of-the-art zero-shot LLM-based manipulation methods: Code as Policies (CAP) [14], VoxPoser [15], and single-agent (SA) [11]. For CAP[2], we build upon its official PyBullet [32] implementation and adapt it for use with RLBench. We used the official implementation of VoxPoser[3] without any modifications. Our single-agent (SA) baseline is a version of "Language Models as Zero-Shot Trajectory Generators" [11] with environment-specific (RLBench or real-world) prompt tuning. Additionally, for fair comparison between the Single Agent (SA) and MALMM, the prompts we provided to both of them were equivalent. The only difference is that in the case of MALMM, the information and the instructions in the Single-Agent prompt are distributed between the *Supervisor*, the *Planner*, and the *Coder* according to their respective roles. All the baselines use *gpt-4-turbo* as an LLM.

[2]https://code-as-policies.github.io/
[3]https://voxposer.github.io/

D. Results for Zero-Shot Evaluation

Table I presents the results for the three baseline methods, along with our proposed MALMM, across 9 different tasks. From the table, we observe that MALMM outperforms all baselines across all 9 tasks, including long-horizon tasks such as *stack blocks* and *empty container*, as well as tasks involving complex shapes, such as *meat off grill* and *rubbish in bin*. Moreover, the Code as Policies is able to generate a successful trajectory for only two tasks. This limited success is because the original Code as Policies implementation relied on few-shot examples to perform well on tasks involving regularly shaped objects. In our evaluation, we replaced these few-shot examples with coordinate definitions and detailed instructions about the functions available for the LLM to call. However, Code as Policies completely failed in this zero-shot setting, which was also reported in [11].

VoxPoser [15], which generates 3D voxel maps for value functions to predict waypoints, successfully generated trajectories for three tasks with good accuracy. For two of these tasks, it was the second-best performing method. However, its performance did not generalize well to other tasks.

Both the single-agent (SA) and our proposed multi-agent framework, MALMM, successfully generated trajectories for all 9 tasks. However, MALMM consistently outperformed the SA approach by using different agents for specific roles, enabling it to generate accurate high-level plans and low-level code while mitigating hallucinations. In addition to the *gpt-4-turbo* experiments, we evaluated MALMM using the open-source LLaMA-3.3-70B [28]; the results are presented in Table I. Although there is a drop in performance compared to *gpt-4-turbo*, MALMM, with LLaMA-3.3-70B, outperforms the existing baselines by a sizable margin.

E. Multi-Agent Ablation

We performed ablation for each of the components (agents) in MALMM in order to evaluate how each of them

TABLE II: Ablation study assessing the impact of different components in MALMM: environment feedback, *Planner* (**P**), *Coder* (**C**), and *Supervisor* (**S**).

Agents	Environment Feedback	P	C	S	Stack Blocks	Empty Container
Single Agent (SA)	✗	✗	✗	✗	0.08	0.12
Single Agent (SA)	✓	✗	✗	✗	0.20	0.36
Multi-Agent (MA)	✓	✓	✓	✗	0.36	0.48
MALMM	✓	✓	✓	✓	0.56	0.64

contributes to the overall performance. We considered two tasks, namely *stack blocks* and *empty container*, and report the results in Table II.

We first analyzed the importance of the intermediate environment feedback. To this end, we considered the Single Agent (SA) setting and removed the environmental feedback provided after each intermediate step. In this setup, LLM generates the full manipulation plan at once and executes it without revisions. As shown in Table II, the Single Agent without environmental feedback exhibits 12% and 24% drop in performance for the *stack blocks* and *empty container* tasks, respectively. By analyzing the failure cases of both methods, we observed that the environmental feedback provided after each intermediate step crucially affected the agent's ability to detect unforeseen situations and recover from failures such as collisions and inaccurate grasping.

We next validated the advantage of the Multi-Agent architecture with separate LLM agents for planning and code generation. As shown in Table II, the Multi-Agent system (MA), consisting of a dedicated *Planner* and *Coder*, demonstrated 16% and 12% performance improvement over SA for the two tasks respectively. This can be attributed to the inherent limitations of LLMs in managing very long context conversations [6]. In the SA setup, where a single LLM is responsible for both the high-level planning and the low-level code generation, the agent must handle an extensive context, particularly for long-horizon tasks. This often leads to errors such as failing to account for collisions with other objects, omitting the input arguments for the predefined functions, using variables before they were initialized, and even forgetting the specified goal. In contrast, the MA system mitigates these issues by dividing the workload among specialized agents. The *Planner* and the *Coder* agents in the MA setting focus on specific roles through specialized prompts and communicate with each other, thus reducing the likelihood of errors and hallucinations, in particular for longer tasks.

Our initial Multi-Agent system pre-defines the cyclic sequence of the *Planner*, the *Coder*, and the *Coder Executor*, see Fig. 3(b), assuming that each agent correctly completes its task. However, hallucinations may occur even within multi-agent systems [33]. For example, the *Coder* may miss the variable initialization resulting in compilation errors or incomplete sub-goal code generation, such as producing code only for approaching the object without grasping it. In such situations, the *Coder* may need to be re-executed in order to correct possible errors before passing the control to the *Planner*. To automate this process, we introduced a *Supervisor* agent that dynamically re-routes the execution process to the next agent based on the input instruction, the entire communication history of all active agents, and the role descriptions of all agents rather than following a fixed sequence. This adaptive approach is at the core of our MALMM framework, and it improves the performance of the dual-agent MA setup by 20% in the *'stack blocks'* task and 16% in the *'empty container'* task, respectively, as shown in Table II.

F. MALMM is Better at Long-Horizon Planning

To validate the effectiveness of MALMM in long-horizon tasks, we created three variations of the *'stack blocks'* task, each with a different number of blocks, and compared the performance of MALMM to the Single Agent setup. The results in Fig. 5 indicate that while the Single Agent setup struggles with stacking 3 and 4 blocks, MALMM substantially outperforms SA, in particular for tasks that require longer planning.

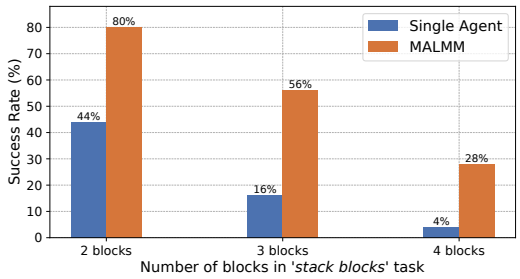


Fig. 5: Comparison of **Single Agent** vs. **MALMM** for variations of the *stack blocks* task that require stacking 2, 3, or 4 blocks on top of each other.

G. Results for Vision-Based Observations

In order to approach real-world settings where direct access to the environment states is not available, we next perform experiments in simulation restricted to vision-based observations in the form of 3D point clouds. We evaluate the performance of MALMM and compare it to the Single Agent setup on three tasks: (*ii*) *put block*, (*ii*) *rubbish in bin*, and (*iii*) *close jar*. Consistently with our previous state-based experiments, the results in MALMM in Table III show sizable improvements over the Single-Agent baseline across all three tasks and confirm the advantage of our proposed Multi-Agent framework.

By comparing the results in Tables I and III we observe degradation in performance when switching to vision-based observations. This can be attributed to inaccuracies of the vision-based estimators such as 3D bounding box detection and grasp estimation. Note that our current vision pipeline makes use of single-view scene observations. A parallel work, Manipulate Anything [34] showed that a relatively straightforward extension to multi-view settings can reduce the impact of occlusions and yield higher accuracy.

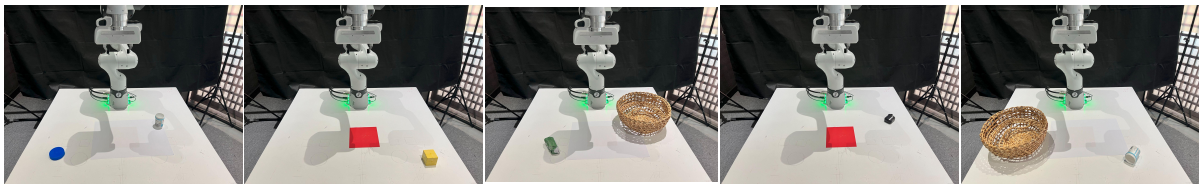


Fig. 6: Illustration of the **five** real-world tasks used in our evaluation.

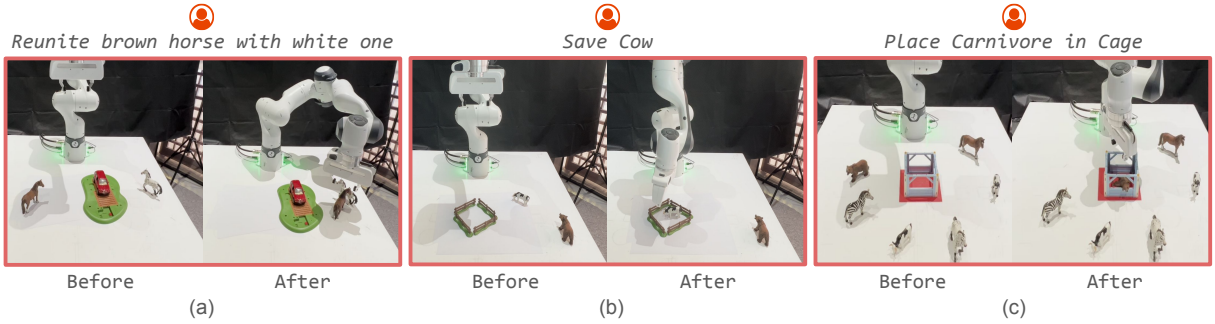


Fig. 7: MALMM performs zero-shot manipulation on three unseen tasks in a real world, each guided by *high-level* user instructions. (a) "*Reunite the brown horse with the white one.*" The environment contains a brown horse, a white horse, a road, and a car; the goal is to place the brown horse near the white horse. (b) "*Save Cow.*" The environment contains a bear, a cow, and a fenced enclosure; the goal is to place the cow inside the enclosure. (c) "*Place Carnivore in Cage.*" The environment contains two horses, two cows, one zebra, a bear, and a cage; the goal is to place the bear inside the cage. In each scenario, the left image shows the initial arrangement and the right image shows the final arrangement after MALMM completes the instructed task.

TABLE III: Comparison of the Single Agent and MALMM in a simulated environment with vision-based observations.

Agents	Close Jar	Put Block	Rubbish in Bin
Single Agent [11]	0.24	0.68	0.40
MALMM	**0.56**	**0.84**	**0.52**

TABLE IV: Comparison of Single Agent and MALMM in a real-world Franka robot arm environment.

Agents	Close Jar	Put Block	Rubbish in Bin	Put Case	Jar in Bin
Single Agent [11]	2/10	3/10	2/10	3/10	3/10
MALMM	**4/10**	**6/10**	**6/10**	**7/10**	**5/10**

H. Results for the Real-World Experiments

For the real-world robotics setup, we evaluated five tasks as shown in Fig. 6: *close jar* (put lid on top of jar), *put block* (place a block in the red target area), *put rubbish in bin* (place rubbish in the bin), *put case* (place an earbuds case in the red target area), and *put jar in bin* (place a jar in the bin) – in ten different initial states, each with both MALMM and the Single Agent. As shown in Table IV, consistently with our simulation results, MALMM outperforms the Single Agent on all five tasks by a sizable margin. It achieved a 40% higher success rate for both *put case* and *rubbish in bin*, 30% higher success rate for *put block*, and 20% higher success rate for *close jar* and *jar in bin*. To further demonstrate zero-shot capabilities of our method, Fig. 7 demonstrates successful performance of MALMM for three new tasks, each defined by *high-level* user instruction.

V. DISCUSSION

A. Limitations

Despite its string advantages over a Single LLM Agent, MALMM has several limitations. First, MALMM relies on three *gpt-4-turbo* agents, making it costly to operate. Using open-source LLMs is possible at the cost of reduced accuracy (cf. Table I). Second, like other LLM-based planners, MALMM depends on manual prompt engineering, which impacts its performance. However, advancements in prompting [5] can reduce these efforts. Finally, MALMM requires accurate bounding box estimation to determine the correct grasp positions, but as the complexity of the objects increases, the bounding boxes alone may not provide enough information for precise grasping. Our experiments with a vision pipeline in simulation and in a real-world scenario suggest that using pretrained grasping and placement models, such as M2T2 [27], could improve the performance for complex manipulation tasks.

B. Conclusion and Future Work

We explored the use of LLM agents for solving previously unseen manipulation tasks. In particular, we proposed the first multi-agent LLM framework for robotics manipulation MALMM and demonstrated its advantages over single-agent baselines. Our method uses task-agnostic prompts and

requires no in-context learning examples for solving new tasks. Extensive evaluations, both in simulation and real-world settings, demonstrated excellent results for MALMM for a variety of manipulation tasks. In future work, we will consider tasks with richer object interactions, and we will extend our experiments to more complex tasks.

REFERENCES

[1] A. Goyal, J. Xu, Y. Guo, V. Blukis, Y.-W. Chao, and D. Fox, "Rvt: Robotic view transformer for 3d object manipulation," *arXiv preprint arXiv:2306.14896*, 2023.

[2] T. Gervet, Z. Xian, N. Gkanatsios, and K. Fragkiadaki, "Act3d: 3d feature field transformers for multi-task robotic manipulation," in *Conference on Robot Learning*. PMLR, 2023, pp. 3949–3965.

[3] J. Gao, B. Sarkar, F. Xia, T. Xiao, J. Wu, B. Ichter, A. Majumdar, and D. Sadigh, "Physically grounded vision-language models for robotic manipulation," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 12 462–12 469.

[4] E. Davis, "Mathematics, word problems, common sense, and artificial intelligence," *Bulletin of the American Mathematical Society*, vol. 61, no. 2, pp. 287–303, 2024.

[5] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. H. hsin Chi, F. Xia, Q. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *ArXiv*, vol. abs/2201.11903, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246411621

[6] A. Maharana, D.-H. Lee, S. Tulyakov, M. Bansal, F. Barbieri, and Y. Fang, "Evaluating very long-term conversational memory of llm agents," *ArXiv*, vol. abs/2402.17753, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:268041615

[7] J. Thomason, S. Zhang, R. J. Mooney, and P. Stone, "Learning to interpret natural language commands through human-robot dialog," in *International Joint Conference on Artificial Intelligence*, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:6745034

[8] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, "Bc-z: Zero-shot task generalization with robotic imitation learning," *ArXiv*, vol. abs/2202.02005, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:237257594

[9] T. Kollar, S. Tellex, D. K. Roy, and N. Roy, "Grounding verbs of motion in natural language commands to robots," in *International Symposium on Experimental Robotics*, 2010. [Online]. Available: https://api.semanticscholar.org/CorpusID:5821491

[10] Y. Wang, Z. Sun, J. Zhang, Z. Xian, E. Biyik, D. Held, and Z. M. Erickson, "Rl-vlm-f: Reinforcement learning from vision language foundation model feedback," *ArXiv*, vol. abs/2402.03681, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:267499679

[11] T. Kwon, N. D. Palo, and E. Johns, "Language models as zero-shot trajectory generators," *IEEE Robotics and Automation Letters*, vol. 9, pp. 6728–6735, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:264289016

[12] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. R. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, N. Brown, T. Jackson, L. Luu, S. Levine, K. Hausman, and B. Ichter, "Inner monologue: Embodied reasoning through planning with language models," in *Conference on Robot Learning*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:250451569

[13] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," *ArXiv*, vol. abs/2306.15724, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259274760

[14] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. R. Florence, and A. Zeng, "Code as policies: Language model programs for embodied control," *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252355542

[15] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," *ArXiv*, vol. abs/2307.05973, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259837330

[16] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. M. O. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia, "Language to rewards for robotic skill synthesis," *ArXiv*, vol. abs/2306.08647, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259164906

[17] S. Bubeck, V. Chandrasekaran, R. Eldan, J. A. Gehrke, E. Horvitz, E. Kamar, P. Lee, Y. T. Lee, Y.-F. Li, S. M. Lundberg, H. Nori, H. Palangi, M. T. Ribeiro, and Y. Zhang, "Sparks of artificial general intelligence: Early experiments with gpt-4," *ArXiv*, vol. abs/2303.12712, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257663729

[18] Z. Mandi, S. Jain, and S. Song, "Roco: Dialectic multi-robot collaboration with large language models," *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 286–299, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:259501567

[19] I. Dasgupta, C. Kaeser-Chen, K. Marino, A. Ahuja, S. Babayan, F. Hill, and R. Fergus, "Collaborating with language models for embodied reasoning," *ArXiv*, vol. abs/2302.00763, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:253180684

[20] C. Huang, O. Mees, A. Zeng, and W. Burgard, "Visual language maps for robot navigation," *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10 608–10 615, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:252846548

[21] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," *ArXiv*, vol. abs/2201.07207, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:246035276

[22] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison, "Rlbench: The robot learning benchmark & learning environment," *IEEE Robotics and Automation Letters*, vol. 5, pp. 3019–3026, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:202889132

[23] D. Surís, S. Menon, and C. Vondrick, "Vipergpt: Visual inference via python execution for reasoning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023, pp. 11 888–11 898.

[24] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," *arXiv preprint arXiv:2211.12588*, 2022.

[25] OpenAI, "GPT-4 technical report," 2023.

[26] L. Medeiros, "Language segment-anything," https://github.com/luca-medeiros/lang-segment-anything, 2024, accessed: 2024-09-12.

[27] W. Yuan, A. Murali, A. Mousavian, and D. Fox, "M2t2: Multi-task masked transformer for object-centric pick and place," in *Conference on Robot Learning*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263629874

[28] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, and A. Y. et al, "The llama 3 herd of models," *ArXiv*, vol. abs/2407.21783, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:271571434

[29] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang, "Autogen: Enabling next-gen llm applications via multi-agent conversation," 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:263611068

[30] "MALMM project webpage," https://malmm1.github.io/.

[31] J. Elsner, "Taming the panda with python: A powerful duo for seamless robotics programming and integration," *SoftwareX*, vol. 24, p. 101532, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711023002285

[32] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," http://pybullet.org, 2016–2021.

[33] S. Banerjee, A. Agarwal, and S. Singla, "Llms will always hallucinate, and we need to live with this," 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:272524830

[34] J. Duan, W. Yuan, W. Pumacay, Y. R. Wang, K. Ehsani, D. Fox, and R. Krishna, "Manipulate-anything: Automating real-world robots using vision-language models," *ArXiv*, vol. abs/2406.18915, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:270764457