

Supplementary Material for Manipulate-Anything

Anonymous Author(s)

Affiliation

Address

email

Abstract: This is the supplementary material for MANIPULATE-ANYTHING. It comprises the detailed implementation of MANIPULATE-ANYTHING prompts, simulation benchmark, and also the real-world experiments setup details. For more detailed videos on both simulation and real-world experiments, refer to our anonymous webpage: manipulate-anything.github.io.

Keywords: Zero-shot manipulation, multimodal language models, multiview state verification, robot skill generation, behavior cloning, robotic manipulation

1 MANIPULATE-ANYTHING implementations

Action Generation Module. We generate each action using either an agent-centric or object-centric approach. For object-centric action generation, we utilize M2T2[1], NVIDIA’s foundational grasp prediction model, for pick and place actions. For 6-DoF grasping, we input a single 3D point cloud from either a single RGB-D camera (in the real world) or multiple cameras (in simulation). The model outputs a set of grasp proposals on any graspable objects, providing 6-DoF grasp candidates (3-DoF rotation and 3-DoF translation) and default gripper close states. For placement actions, M2T2 outputs a set of 6-DoF placement poses, indicating where the end-effector should be before executing a drop primitive action based on a VLM plan. The network ensures the object is stably positioned without collisions. We also set default values for `mask_threshold` and `object_threshold` to control the number of proposed grasp candidates. After proposing a list of template grasp poses, we use QWen-VL[2] to detect the target object by prompting the current image frame with the target object’s name, translated into Chinese using a machine translation model [3]. This detection is applied to all re-rendered viewpoints or viewpoints from different cameras. We then concatenate these frames into a single image, annotating each sub-image with a number at the top right corner. Next, we call the GPT-4V API with few-shot demonstrations and the task goal to prompt GPT-4V to output the selected number of viewpoints that provide the most unobstructed views for sampling the grasp pose to achieve the sub-goal. Using the selected viewpoint, we execute the grasp by moving the end-effector to the sampled grasp pose via a motion planner.

For agent-centric action generation, we first perform the same steps of viewpoint selection. Using the selected viewpoint, a few demonstration examples, and the sub-goal, we prompt GPT-4V to generate an action function with code snippets that include the necessary code to perform a delta-action on the current robot pose. We then execute this by moving the end-effector based on these delta changes. This process is iterated until we obtain the most desirable code snippet function for the given sub-goals, which is then appended to a skill library for future use.

Sub-goal Verification Module The sub-goal verification module helps with error recovery by ensuring all potential attempts at resolving the current step action have been tried. With the temporary goal state obtained by the action generation module, we use multi-viewpoints to sample the optimal viewpoint for answering the verification condition generated for the given sub-goal during the task plan generation phase. Using the same viewpoint selection method as in the Action Generation Module, we obtain the optimal view and then perform a two-step sequence rollout of frames: one from the current frame at this viewpoint and another from the previous action step. We concatenate

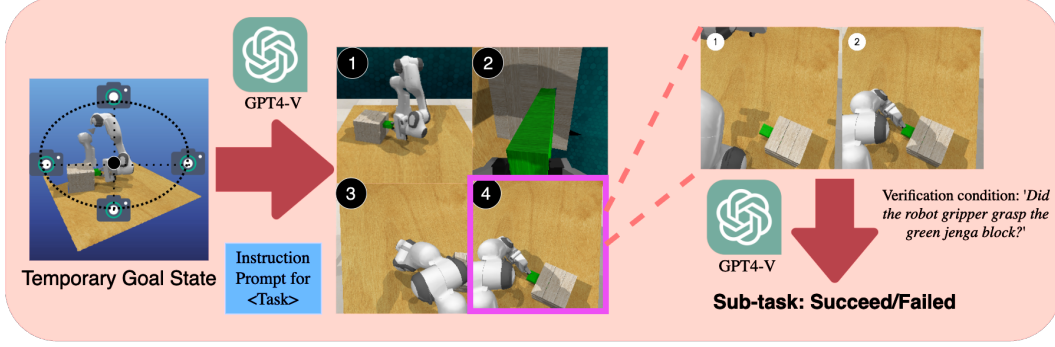


Figure 1: **Sub-goal Verification Module.** We used viewpoint selection similar to action generation, to find the optimal viewpoints, and roll out the two step sequences of the previous and current frames for prompting the verification condition.

these two frames, annotate them with numbers to indicate their temporal relation, and use this image to prompt GPT-4V to check if the verification condition is fulfilled as shown in Fig. 1. If the answer is Yes, we proceed to the next sub-goal. If the answer is No, we resample new viewpoints, generate new actions, and reattempt the entire sub-goal with a different seed.

2 Simulation experiments

Simulation Setup. All the simulated experiments use a four-camera setup as illustrated in Fig. 3. The cameras are positioned at the front, left shoulder, wrist, and right shoulder. All cameras are static, except for the wrist camera, which is mounted on the end effector. We did not modify the default camera poses from the original RLBench [4]. These poses maximize coverage of the entire table, and we use a 256 x 256 resolution for better input to the VLMs.

Task Details. We describe in detail each of the 12 tasks for simulation evaluation, both for trained policies and zero-shot methods, along with their RLBench variations and success conditions. We have made some modifications to the original tasks to enhance the detection rate by Code-As-Policies and VoxPoser.

2.1 put block

Filename: put_block.py

Task: Pick up the green block and place it on the red mat.

Success Metric: The success condition on the red mat detects the target green block.

2.2 close box

Filename: close_box.py

Task: Close the box.

Success Metric: The revolute joint of the specified handle is at least 60° off from the starting position.

2.3 open box

Filename: open_box.py

Task: Open the box.

Success Metric: The revolute joint of the specified handle is at least 60° off from the starting position.

66 **2.4** play jenga

67 **Filename:** play_jenga.py

68 **Task:** Pull out the green jenga block.

69 **Success Metric:** The green jenga block is out of its pre-defined location.

70 **2.5** open jar

71 **Filename:** open_jar.py

72 **Task:** Uncap the green jar.

73 **Success Metric:** The green jar is out of its pre-defined capped location.

74 **2.6** pickup cup

75 **Filename:** pickup_cup.py

76 **Task:** Pick up the red cup.

77 **Success Metric:** Lift up the red cup above the pre-defined location.

78 **2.7** take umbrella

79 **Filename:** take_umbrella_out_of_stand.py

80 **Task:** Pick up the umbrella out of the umbrella stand.

81 **Success Metric:** Lift up the umbrella out of the umbrella stand.

82 **2.8** sort mustard

83 **Filename:** sort_mustard.py

84 **Task:** Pick up the yellow mustard bottle, and place it into the red container.

85 **Success Metric:** The yellow mustard bottle inside red container.

86 **2.9** open wine

87 **Filename:** open_wine.py

88 **Task:** Uncap the wine bottle.

89 **Success Metric:** The wine bottle cap is out of its original position.

90 **2.10** lamp on

91 **Filename:** lamp_on.py

92 **Task:** Turn on the lamp.

93 **Success Metric:** The lamp light up.

94 **2.11** put knife

95 **Filename:** put_knife_on_chopping_board.py

96 **Task:** Pick up the knife and place it onto the chopping board.

97 **Success Metric:** Knife placed on chopping board.

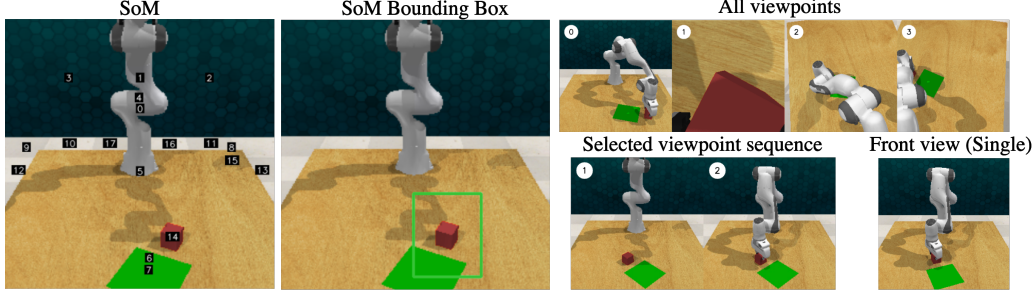


Figure 2: **Evaluation of visual prompting.** We systematic evaluate 5 different visual prompting techniques, and found that selected viewpoint sequence yields the highest performance.

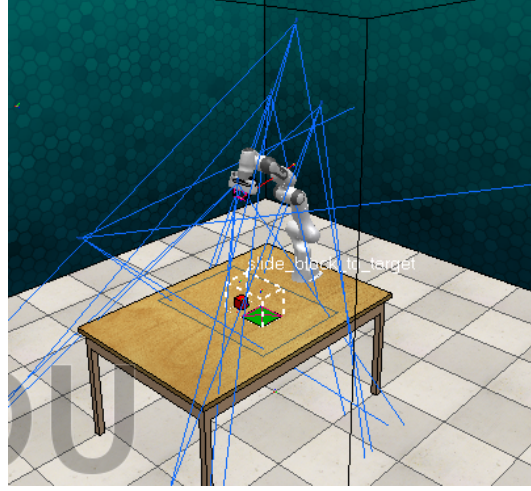
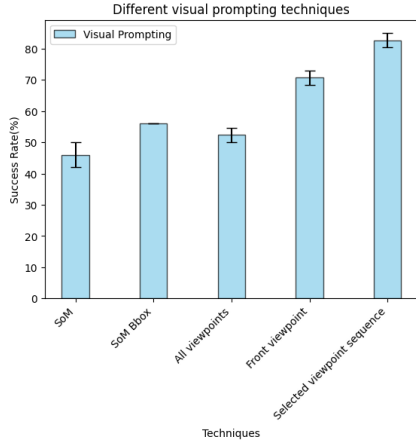


Figure 3: **Results for visual prompting techniques (Left).** We reported the various results for different visual prompting technique decision, and reported that selected viewpoint sequence yield the best performance. **Simulation scene setup (Right).** We leverage 4 different camera for evaluation.

98 **2.12 pick & lift**

99 **Filename:** pick_and_lift.py

100 **Task:** Pick up the red cube.

101 **Success Metric:** The red cube is lifted up.

102 3 Real-world experiments

103 3.1 Robot hardware setup

104 The real-robot experiments use a Franka Panda manipulator with a parallel gripper. For perception,
 105 we use a Kinect-2 RGB-D camera mounted on a tripod, at an angle, pointing towards the tabletop.
 106 Kinect-2 provides RGB-D images of resolution 512×424 at 30Hz. The extrinsic between the camera
 107 and robot base-frame are calibrated with the easy hand-eye package. We use an ARUCO AR marker
 108 mounted on the gripper to aid the calibration process, as shown in Figure 4.

109 3.2 Additional real-world everyday manipulation tasks

110 Beyond the five real-world experiments used for systematically evaluating MANIPULATE-ANYTHING,
 111 we also have additional real-world demonstrations generated in a zero-shot manner via MANIPULATE-

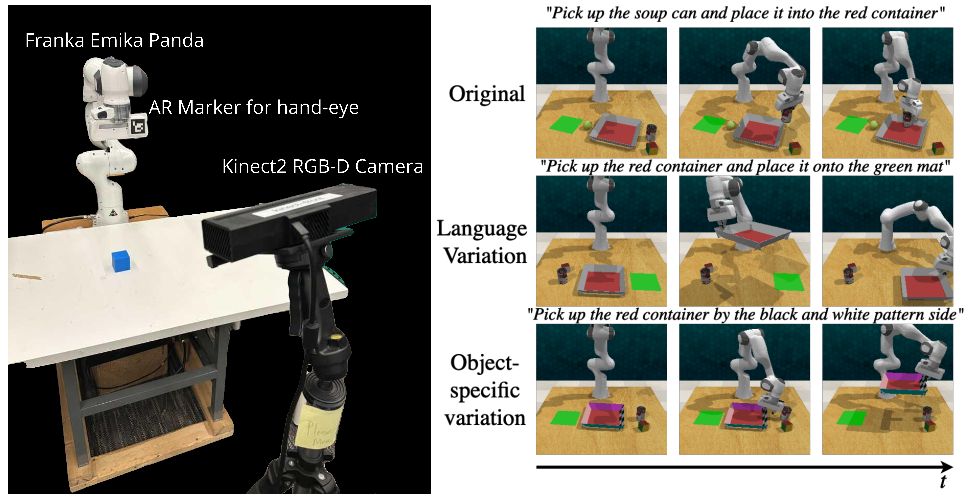


Figure 4: **Real-world experiment setup (Left).** We set up the real-world using this configuration. **Robustness and generalization evaluation.** We evaluated MANIPULATE-ANYTHING against VoxPoser for capability in generalizing to different language instructions and also object-specific manipulation.

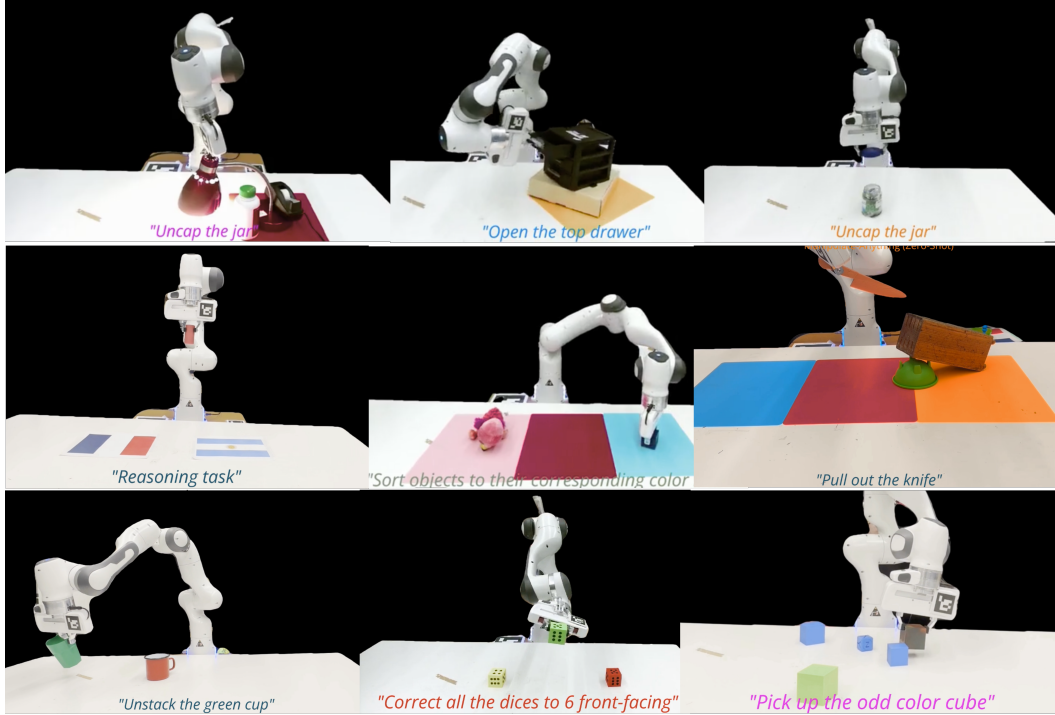


Figure 5: **More real-world experiments.**

112 ANYTHING. These demonstrations cover a range of tasks, from reasoning tasks to more precise
 113 everyday tasks. All of the tasks can be seen in Fig. 5.

114 4 Additional Ablation Studies

115 We conducted two main set of ablation studies, we first look at how different visual prompting works
 116 for sub-goal verification, and then we further evaluated MANIPULATE-ANYTHING’s robustness and
 117 generalization to language instructions in another set of experiments.

118 For evaluating different visual prompts for sub-goal verification on the put_block task, we employed
119 the following methods: 1) Set-of-Mark [5] on a single view, 2) Set-of-Mark with bounding box
120 annotation, 3) Concatenated all viewpoints, 4) Front view only, and 5) Selected viewpoint sequence
121 as shown in Fig. 2. We observed that the selected viewpoint sequences were the most effective in
122 achieving correct sub-goal verification, obtaining the highest success rate as shown in Fig. 3.

123 We further evaluated the generalization capabilities of our model in terms of object-specific manipu-
124 lation and robustness to changes in language instructions. For language instruction variations, we
125 altered the instructions for the same scene and found that MANIPULATE-ANYTHING outperforms
126 VoxPoser by 60% over 25 episodes. For object-specific variations, where instructions targeted specific
127 parts of objects, MANIPULATE-ANYTHING outperformed VoxPoser by 16%, as shown in Fig. 4.

128

129 References

- 130 [1] W. Yuan, A. Murali, A. Mousavian, and D. Fox. M2t2: Multi-task masked transformer for
131 object-centric pick and place, 2023.
- 132 [2] J. Bai, S. Bai, S. Yang, S. Wang, S. Tan, P. Wang, J. Lin, C. Zhou, and J. Zhou. Qwen-vl: A
133 frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*,
134 2023.
- 135 [3] J. Tiedemann, M. Aulamo, D. Bakshandaeva, M. Boggia, S.-A. Grönroos, T. Nieminen, A. Ra-
136 ganato, Y. Scherrer, R. Vazquez, and S. Virpioja. Democratizing neural machine translation with
137 opus-mt. *Language Resources and Evaluation*, pages 1–43, 2023.
- 138 [4] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark &
139 learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.
- 140 [5] J. Yang, H. Zhang, F. Li, X. Zou, C. Li, and J. Gao. Set-of-mark prompting unleashes extraordi-
141 nary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023.