Appendix File for REMAC: Self-Reflective and Self-Evolving Multi-Agent Collaboration for Long-Horizon Robot Manipulation

Anonymous submission

***Abstract*— This is the appendix file for REMAC: Self-Reflective and Self-Evolving Multi-Agent Collaboration for Long-Horizon Robot Manipulation. The project code, demos, and appendix are available at the website https://reamac-repo.github.io/remac-project-page/.**

I. Baseline Comparism

As we aim to handle tasks with certain reasoning difficulty, as shown in the task suite description, there is not much comparable work for our reflection-based evolutionary framework. DoReMi [1] uses the idea of real-time detection, which is inconsistent with our condition-checking and self-evolvement framework. SMART-LLM [2] adopts a three-stage multi-robot task planning and allocation method, which lacks relfection ability, and has poor performance on reasoning-difficult tasks. CAPE [3] adopt similar pre-condition check method, but has no evolvement ability, while RELFECT [4] has reflecetive ability but not pre-condition check ability, which are added to our baseline comparison in Figure 6.

II. Why to choose reason models like DeepSeek-R1 for initial planning

We tried to set 3 different types of goal prompt which has the same actual meaning, and conduct experiments on them using different LLMs:

1) Description 1: word order align with subtask order
2) Description 2: word order misalign with subtask order
3) Description 3: the simplest description

For example, in the experiment OpenMicrowavePnP, the 3 goal prompt are as follows (the microwave is initially closed, and is also stated in the scene description):

1) Description 1: open microwave door, pick up the vegetable from the counter and place it into the microwave
2) Description 2: pick up the vegetable from the counter and place it into the microwave; the microwave needs to be opened first.
3) Description 3: place the vegetable into the microwave.

We examine the initial plan outcome for different models, and compare the responses of models with and without reasoning capabilities. The results are shown in Table I.

The results indicated that the planning of large language models is easily affected by the goal word order, which appears in models with and without reasoning ability. Moreover, the reasoning-capable models like DeepSeek-R1 are more likely to leverage information provided by the reflection context, while the gpt-4o model tend to ignore the reflection tips and stick to wrong answer. So we use a reasoning model (DeepSeek-R1) to reflect and generate initial plan, and use a non-reasoning model (gpt-4o) to serve as condition checkers and plan adjusters for time saving. Although Grok3, o3, and Qwen3 have a better reflect success rate across different reasoning models in Table 2, we chose DeepSeek-R1 for its low cost.

III. Further Description for Charts and Tables

Figure 4 is a visualization of Table 1, showing a clearer trend that the task success rate is improved in BASE→CC→RE and maintained in RE→REMAC. In CC→RE→REMAC, the efficiency is also improved. In Figure 4, a dot means a single rollout, and bar means the average value of the corresponding performance.

In Table 1, the value indicates the reflection success rate for a certain LLM model to relfect on pre-condition check results in single turn, while in Figure 4, the values indicate the rollout success rate for a whole interaction.

IV. Implementation and Prompt Details

We open-sourced all the code and demos on our website.

Here we elaborate on the composition of pre- and post-conditions check module, and the former includes multiple parts: pre-conditions generation submodule, pre-conditions checking submodule, pre-conditions analysis submodule, high-level plan adjustment submodule, and high-level plan reflection submodule.

1) Pre-conditions generation submodule: use an LLM to generate pre-conditons for VLM to check the observation with. Return multiple strings for checking.
2) Pre-conditions checking submodule: call a VLM to check whether pre-conditions are satisfied, given side and wrist robot images. Return multiple boolean values to indicate the checking result.
3) Pre-conditions analysis submodule: call an LLM to extract whether the pre-conditions are satisfied. If not, analyze the results and output the reason why they are not satisfied and the recommended next steps.
4) High-level plan adjustment: given previous high-level plan and pre-conditions analysis result, adjust the original high-level plan, and instruct the robot to execute the first subtask (command).

TABLE I: Initial Plan Success Rate for different models and settings. We use gpt-4o [5] and DeepSeek-R1 [6] as representatives of models w/o and w/ reasoning ability. The result is the average of 10 trials each from 4 tasks. Description 1 - word order align with subtask order, Description 2 - word order misalign with subtask order, Description 3 - the simplest description.

	gpt-4o	gpt-4o w/ reflection	DeepSeek-R1	DeepSeek-R1 w/ reflection
Description 1	65%	82.5%	72.5%	87.5%
Description 2	0%	17.5%	22.5%	77.5%
Description 3	0%	10%	17.5%	75%

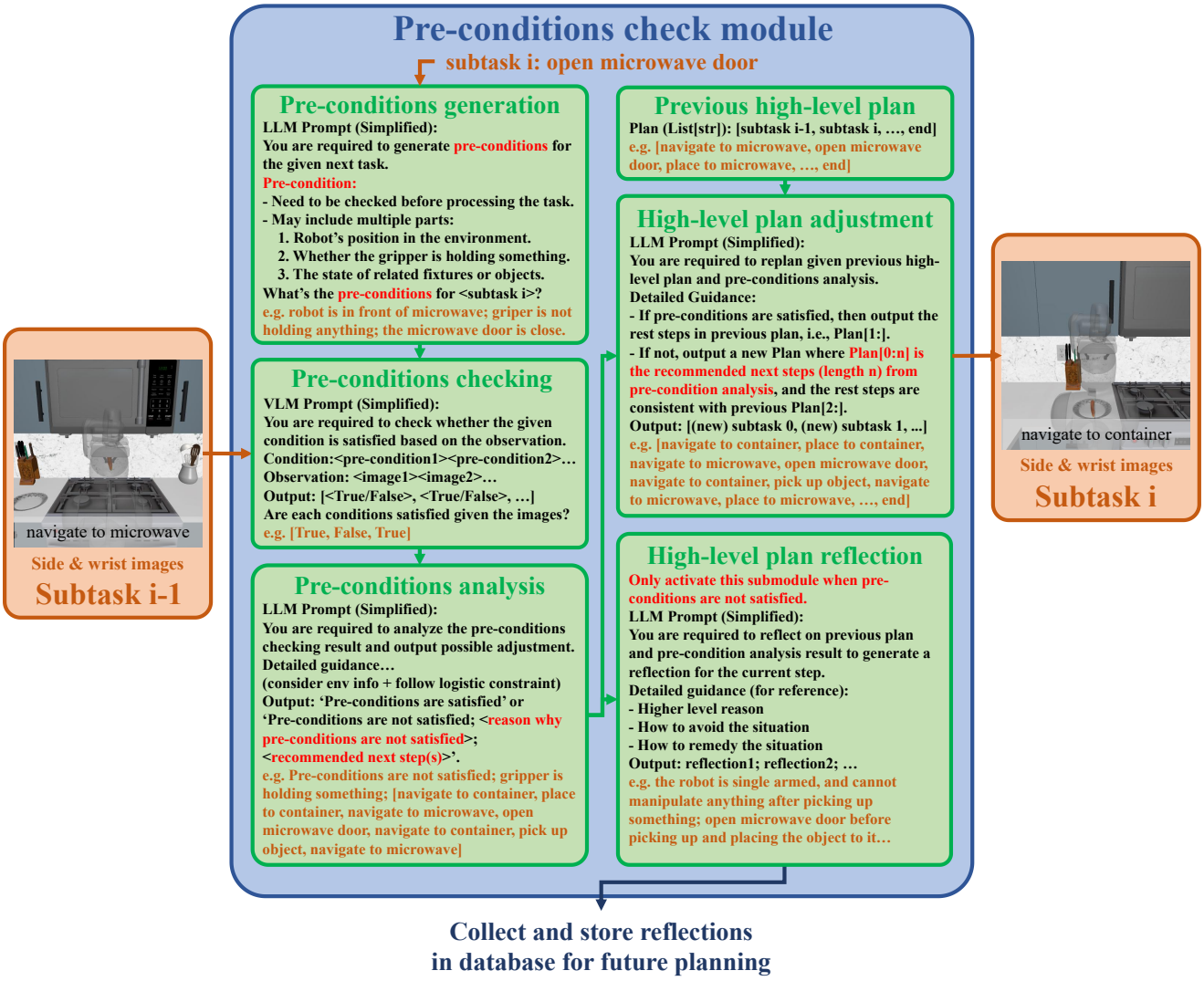

Fig. 1: Simplified prompt example for pre-conditions check module, including pre-condition generation, checking, analysis, plan adjustment and reflection submodule. Please zoom in for detailed example. The post-condition check module is almost identical except for the missing reflection module.

5) High-level plan reflection: based on pre-conditions analysis result, further reflect on the ignorance in the environment and logical constraints in the initial high-level plan. The reflection are formalized into strings and will be stored in the database and used in the future.

Figure 1 is an example for a pre-conditions check module, containing simplified prompts and output examples of each submodule. The original current subtask is 'open microwave door', and the subtask after plan adjustment is 'navigate to container'. Please check the Figure 1 for more details.

For VLM checking, few-shot prompting (about 5 prompts) is used for generating pre-conditions and post-conditions.

This information cannot be provided in the task planning as well, because before execution, the LLM cannot accurately foresee potential errors in the robot-environment interaction process. However, after one trial, the robot could capture potential logical relationships and potential risks more accurately for the same task, thus benefiting the next

initial planning.

V. Real World Deployment

We performed a real-world experiment using an extended version of the OpenCabinetPnP benchmark. The task involves opening a closed microwave, placing food inside, closing the door, and then adding seasoning. Our proposed framework successfully completed this entire multi-step task without any additional modifications. This execution on a more complex and previously unseen task demonstrates the generalization ability of our method, proving it can handle real-world complexities such as sensor noise and unmodeled contact dynamics in a long-horizon scenario with acceptable latency. A detailed video is available on our project page.

VI. Database Storage

We store interaction-related informations in the database, namely task description, task goal, initial plan, action list, pre-check results, post-check results, success flag, and reflection.

1) Task description: a string to provide basic information for the environment, containing usable tools and objects.
2) Task goal: a string to allocate a specific task for robots, always simple and short.
3) Initial plan: a plan that is initially generated by the LLM, may contain wrong action or sequence.
4) Action list: list of actions that are executed in the interaction, may not be the same as the initial plan.
5) Pre-check results: pre-check results for each action executed.
6) Post-check results: post-check results for each action executed.
7) Success flag: a boolean value indicating whether the task is completed.
8) Reflection: reflection string after a task is done, always contains analysis for pre-condition check failure and overall process.

After one iteration is done, the task description, task goal, and the reflection are provided to the future iteration.

VII. Task Planning Safety

We introduce pre-conditions checking mechanism to avoid unsafe actions, which serve as safeguards before execution to a large extent. Moreover, we have post-conditions checking mechanism to ensure the successful execution of the subtask.

What is need to be mentioned is that, the planning LLM (DeepSeek-R1) rarely outputs unfeasible actions when given a clear task description in our experiments.

VIII. Training Details

A. Dataset Format

The dataset used in this paper to train the diffusion policy for primitive actions was generated using the **MimicGen** [7] tool. The format of this dataset follows the standard from the **RoboCasa** [8] research and is based on the `robomimic .hdf5` file convention.

The core structure of each `.hdf5` dataset file is organized as follows:

- `data`: This group contains the environment metadata and all demonstration data.
  - `env_args` (attribute): Records metadata information about the task environment.
  - `demo_<n>`: Stores all data for demonstration *n*.
    - `model_file` (attribute): The XML string corresponding to the MuJoCo MJCF model.
    - `ep_meta` (attribute): Metadata for a single episode, such as the natural language description of the task, scene information, object information, etc.
    - `actions`: Environment actions, ordered by time. The shape is (N, A), where N is the length of the trajectory, and A is the dimension of the action space.
    - `action_dict`: A dictionary that splits actions by fine-grained components (e.g., end-effector position, rotation, gripper state, etc.).
    - `obs`: A dictionary containing various observation data, such as images, proprioception, etc.
    - `states`: Raw, low-level MuJoCo state vectors, ordered by time. This data is used only for replaying demonstrations and is **not to be used for policy learning**.
- `mask`: This group contains filter key metadata used to split the dataset in different ways (e.g., by specific skills or outcomes).

B. Training Method

Our experimental setup follows the standard configuration used in the **robomimic**[9] framework.

a) Model Architecture.: Our policy network processes multi-modal inputs, including visual and proprioceptive state observations. For visual processing, we employ a ResNet-18 backbone that encodes 84×84 pixel images into a 64-dimensional feature vector, using a Spatial Softmax layer for pooling. For the policy itself, the actor's output head is a Gaussian Mixture Model. The actor network is an MLP with two hidden layers of size 400 and 300, respectively, using ReLU activation functions.

b) Training Hyperparameters.: We train our model using a Behavior Cloning objective from the robomimic suite. We use the AdamW optimizer with an initial learning rate of 1×10^{-4} and a weight decay of 1×10^{-5}. The learning rate is decayed using a cosine annealing scheduler. The model is trained for 2000 epochs with a batch size of 256.

References

[1] Y. Guo, Y.-J. Wang, L. Zha, and J. Chen, "Doremi: Grounding language model by detecting and recovering from plan-execution misalignment," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 12 124–12 131.

[2] S. S. Kannan, V. L. Venkatesh, and B.-C. Min, "Smart-llm: Smart multi-agent robot task planning using large language models," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 12 140–12 147.

[3] S. Sundara Raman, V. Cohen, D. Paulius, I. Idrees, E. Rosen, R. Mooney, and S. Tellex, "CAPE: Corrective Actions from Precondition Errors using Large Language Models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, (In Review).

[4] Z. Liu, A. Bahety, and S. Song, "Reflect: Summarizing robot experiences for failure explanation and correction," *arXiv preprint arXiv:2306.15724*, 2023.

[5] A. Hurst, A. Lerer, A. P. Goucher, A. Perelman, A. Ramesh, A. Clark, A. Ostrow, A. Welihinda, A. Hayes, A. Radford *et al.*, "Gpt-4o system card," *arXiv preprint arXiv:2410.21276*, 2024.

[6] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.

[7] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox, "Mimicgen: A data generation system for scalable robot learning using human demonstrations," *arXiv preprint arXiv:2310.17596*, 2023.

[8] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu, "Robocasa: Large-scale simulation of everyday tasks for generalist robots," *arXiv preprint arXiv:2406.02523*, 2024.

[9] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín, "What matters in learning from offline human demonstrations for robot manipulation," *arXiv preprint arXiv:2108.03298*, 2021.