
Empowering LLM Agents with Zero-Shot Optimal Decision-Making through Q-learning

Jiajun Chai, Sicheng Li, Yuqian Fu, Yuanheng Zhu, Dongbin Zhao

Institution of Automation, Chinese Academy of Sciences

School of Artificial Intelligence, University of Chinese Academy of Sciences

{chaijiajun2020, lisicheng2024, fuyuqian2022, yuanheng.zhu, dongbin.zhao}@ia.ac.cn

Abstract

Current Large language model (LLM) agents succeed in making zero-shot decisions but struggle to make optimal decisions, as they rely on pre-trained probabilities rather than maximizing expected future rewards. In contrast, agents trained via reinforcement learning (RL) could make optimal decisions but require extensive data. We develop an algorithm that combines the zero-shot capabilities of LLMs with the optimization of RL, referred to as the **Model-based LLM Agent with Q-Learning (MLAQ)**. MLAQ employs Q-learning to derive optimal policies from transitions within memory. Unlike RL agents, MLAQ constructs an LLM-based imagination space, where a UCB variant generates imaginary data through interactions with the LLM-based world model to derive zero-shot policies. This approach achieves a sub-linear regret bound, as guaranteed by our theorem. Moreover, MLAQ employs a mixed-examination mechanism to further enhance the quality of imaginary data. We evaluate MLAQ on benchmarks that present significant challenges for existing LLM agents. Results show that MLAQ achieves a optimal rate of over 90% in tasks where other methods struggle to succeed. Additional experiments are conducted to reach the conclusion that introducing model-based RL into LLM agents shows significant potential in optimal decision-making. Our website is available at link.

1 Introduction

Most existing large language model (LLM) agents make decisions through prompt engineering (Wei et al., 2022; Mandi et al., 2023), failing in maximizing expected future rewards. Recent studies start to utilize the Markov Decision Process (MDP) framework for planning (Hao et al., 2023). These LLM agents understand the environment through a domain description in natural language (Yao et al., 2023a). In this work, a **domain** denotes an environment, while a **task** entails an original-target state pair. Existing MDP-based LLM agents always rely on a task-specific replay buffer (Zhang et al., 2024d) and techniques such as Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006). Although some approaches leverage imaginary interactions between an LLM-based **World Model** and **Basic Policy** for zero-shot decision-making (Hao et al., 2023), they still require environmental tools, such as dynamics (Zhao et al., 2024), available actions (Ding et al., 2023), limiting their generality. Moreover, since these agents do not perform RL-based optimization on domain-specific **Memory**, they do not establish a complete RL framework to maximize expected future rewards.

In this work, we empower LLM agents with zero-shot optimal decision-making ability by combining advantages of the LLM and RL. As shown in Figure 1, we introduce **Model-based LLM Agent with Q-Learning (MLAQ)**, an LLM agent framework with three-fold contributions. (1) MLAQ integrates a Q-planner, memory, and imagination space, allowing LLM-based imagination to minimize reliance on environment while utilizing Q-learning to maximize expected future rewards. (2) An MCTS-style planning approach is proposed to perform LLM-based imaginary interactions without any environ-

mental tools, while ensuring a sub-linear regret bound. (3) A mixed-examine mechanism (LLM-based self-examine and env-examine) is proposed to improve the quality of imaginary transitions.

We evaluate MLAQ on well-known benchmarks for LLM agents, which require optimal decision-making for long horizons. There is no existing LLM agent has obtained the optimal policy, while MLAQ achieves over 90% optimal / success rate across most difficulty. The comparison with existing methods fully demonstrates MLAQ’s superior performance in optimal decision-making. In conclusion, integrating model-based RL with LLM agent enables zero-shot optimal decision-making.

2 Background

MDP-based framework. An MDP can be defined as $\mathcal{U} = \{\mathbb{S}, \mathbb{A}, \mathbb{T}, \mathbb{R}, \gamma\}$, where s_t is the state in state space \mathbb{S} at timestep t , and a_t is the action in action space \mathbb{A} , $\mathbb{T}(s_{t+1}|s_t, a_t)$ is dynamics function, \mathbb{R} is reward function, and γ is discount factor (Sutton and Barto, 2018). An agent uses policy π to make decisions $a_t \sim \pi(\cdot|s_t)$. The state value is the expected returns $V^\pi(s_t) = \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k r_{t+k}]$.

LLM-based framework for an agent. Existing approaches construct the above MDP using natural language (Yao et al., 2023a). An LLM-based world model $\hat{\mathbb{T}}(s_{t+1}|s_t, a_t; \tau)$ is utilized to approximate the dynamics function, and an LLM-based basic policy $\pi(\cdot|s_t; \tau)$ is utilized to interact with the world model or environment, where τ is a the domain description in nature language (Hao et al., 2023).

Q-learning. Q-learning is a traditional RL algorithm, in which the Q-value indicates the expected return obtained by executing a under s . The update rule of the Q function is:

$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a'} Q_k(s_{t+1}, a') - Q_k(s_t, a_t) \right), \quad (1)$$

where α is the learning rate and k is the interaction index.

Multi-Armed Bandit. Given state s with m available actions $\delta(s)$, the successive plays on action i yields (i.i.d.) Q-values $Q(s, i)$ (Auer et al., 2002). The Upper Confidence Bound (UCB) (Auer et al., 2002) could achieve a sub-linear regret bound by selecting actions through the following rule:

$$a^* = \arg \max_{a \in \delta(s)} \text{UCB}(s, a) = \arg \max_{a \in \delta(s)} \left[V(c(s, a)) + w \sqrt{N(s)/N(c(s, a))} \right], \quad (2)$$

where $c(s, a)$ is the child node of applying a in s , and w is a coefficient.

3 Method

MLAQ enables LLM agents with zero-shot optimal decision-making capabilities. While integrating RL-based optimization into LLM agents is straightforward, the key challenge lies in preserving their zero- or few-shot abilities when incorporating RL-based optimization within LLM agents.

3.1 Optimizing LLM Agent with Q-Planner

In this section, we illustrate how MLAQ uses RL to achieve zero-shot optimal decision-making under the LLM-based framework. As shown in Figure 1, MLAQ contains a memory module, a Q-Planner, and an LLM-based imagination space. The RL-based optimization is achieved by Q-Planner, which utilizes the transitions (s, a, r, s') in the memory module to optimize the agent through Q-learning. These transitions are generated from both environmental and imaginary interactions. The overall algorithm is detailed in Appendix G.1.

Q-Planner. It serves as the core of the decision-making process, utilizing the optimal decision-making advantage of RL to output the action with the highest Q-value for a given state, where these values are derived from Q-learning iterations in Eq. (1). The Q-learning is not applied directly to the

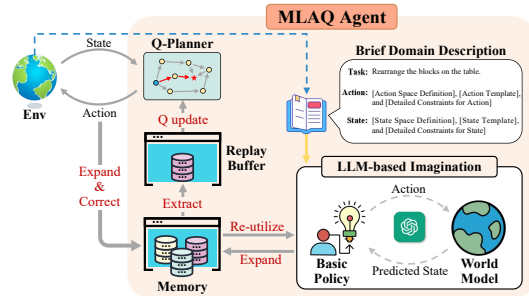


Figure 1: The overall model-based Q-learning framework for MLAQ.

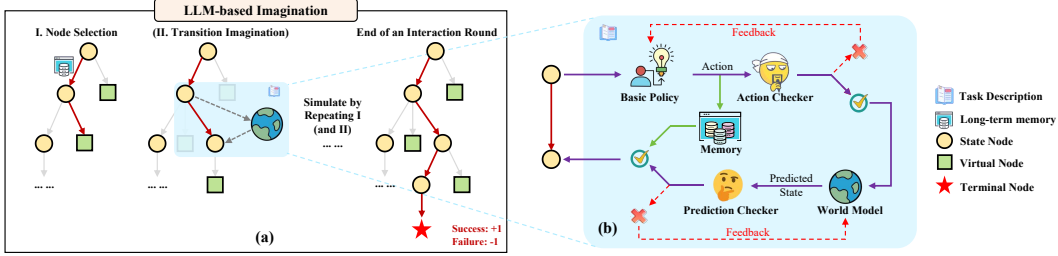


Figure 2: (a) Imaginary interaction process, which expands the memory and replay buffer through two phases. (b) Imagine a new transition with self-examine mechanism and memory.

domain-specific memory \mathcal{M} but rather to a task-specific replay buffer \mathcal{D} extracted from it, aiming to reduce the exploration space in imaginary interactions.

Replay Buffer. When a new task starts, a Q function $Q(\cdot, \cdot)$ is obtained by performing Q-learning on \mathcal{M} , and the initial replay buffer stores the transitions along the optimal trajectory (if exists) produced by $Q(\cdot, \cdot)$. If this trajectory does not exist or is demonstrated to be non-optimal, the agent keeps exploring to expand the replay buffer until the best trajectory is demonstrated to be optimal.

Imaginary Interaction. As shown in Figure 1, an LLM-based basic policy and world model are established to perform imaginary interactions. Therefore, MLAQ could preserve the zero-shot decision-making capabilities of LLM agents while satisfying the data requirements of Q-learning.

Environmental Interaction. MLAQ agent obtains the current state and domain description τ from the environment, explores the imagination space to derive the optimal policy using the Q-planner, and then outputs actions to interact with the environment.

3.2 LLM-based Imagination for MLAQ

This section presents an MCTS-style planning method that utilizes only LLMs to generate imaginary transitions, thereby efficiently expanding the memory and replay buffer. Given a new task, MLAQ performs multiple **imagination rounds**, with each consisting of imaginary interactions transitioning from the original state to the target state. An imagination round comprises two major phases: node selection and transition imagination. Please refer to Appendix G.2 for pseudo-codes.

Node selection. This phase involves balancing exploration and exploitation through a novel selection rule. In LLM agents, since the ground-truth available actions $\delta(s)$ are unknown, MLAQ begins with an empty set $\hat{\delta}(s)$ and introduce the concept of virtual nodes to expand $\hat{\delta}(s)$. Within the replay buffer, a actual (state) node is created based on a state s , and a virtual node is appended as its first child node. Then, MLAQ iteratively leverages LLMs to explore and expand it. This process could be models as:

A Variant of Multi-Armed Bandit. In a scenario with m available actions ($a = \{1, 2, \dots, m\}$) in state s , initially only the virtual node y can be selected. Selecting y enables the least-indexed child node in a to be selected, while selecting any other node involves estimating its Q-value.

When y is selected for the j -th times, i.e., $|\hat{\delta}(s)| = j$, the actual nodes $\{1, 2, \dots, j\}$ can be selected by the planning policy. Thus, we propose a UCB variant by modifying its selection rule as follows:

$$a^* = \arg \max_{a \in \hat{\delta}(s)} \text{vUCB}(s, a) = \arg \max_{a \in \hat{\delta}(s)} [V(c(s, a)) + f(N(s), N(c(s, a)))] , \quad (3)$$

where $f(N(s), N(c))$ is the confidence bound, defined as $w\sqrt{N(s_t)/N(c(s_t, a))}$ for actual nodes and $g(|\hat{\delta}(s_t)|)\sqrt{N(s_t)/|\hat{\delta}(s_t)|}$ for virtual nodes. $g(x)$ is a coefficient function, which are discussed in the Appendix D, where we also provide Theorem 2 to demonstrate its sub-linear regret bound. MLAQ turns to **transition imagination** if virtual nodes are selected. Otherwise, it iterates until reaching success states (target states) or failure states (LLM determines no actions are available).

Transition imagination. This phase generates an imaginary transition. Beginning from s_t , we mark the existing actions in $\hat{\delta}(s_t)$ as forbidden, where the specific prompt can be found in Appendix H. We then instruct the basic policy to output an action a_t excluding forbidden actions to expand $\hat{\delta}(s_t)$. If the pair (s_t, a_t) could be queried in \mathcal{M} , we directly **re-utilize** the stored next state $c(s_t, a_t)$. Otherwise,

Table 1: The optimal rate of experimental methods in BlocksWorld domain.

Methods	2-step	4-step	6-step	8-step	10-step	12-step
CoT (Wei et al., 2022)	0.22	0.14	0.02	0.02	0.00	0.00
REX (Murthy et al., 2023)	0.80	0.45	0.25	-	-	-
RAFA (Liu et al., 2023)	-	0.97	0.75	-	-	-
RAP (Hao et al., 2023)	0.67	0.76	0.74	0.48	0.17	0.09
MLAQ	1.00	1.00	1.00	0.97	0.93	0.90

we employ the LLM-based world model to predict $s_{t+1} \sim \hat{\mathbb{T}}(s_t, a_t; \tau)$. Since we **cannot confirm** if the available actions are exhausted, the virtual node always remains as a child node. Finally, the reward is also obtained by LLMs, following a sparse reward setting (+1 for success and -1 for failure).

3.3 Mixed-Examination for Imaginary Transitions

We use the mixed-examination to improve the quality of imaginary data without environmental tools.

Env-examination. When using domain descriptions to understand the environment for LLMs, both descriptions and LLM’s comprehensions could be incomplete, leading to a gap between the world model and true dynamics. Thus, during environmental interactions, MLAQ corrects the transitions in memory based on environmental transitions and records errors to refine the world model’s prompt.

Self-examination. Considering the **hallucinations** inherent in LLMs, MLAQ adopts an LLM-based action checker and prediction checker to check the outputs of the basic policy and world model as shown in Figure 2 (b). Current models (e.g., GPT-4) already possess the ability to improve output accuracy through self-examination, which will be validated in subsequent experiments.

4 Experiments

4.1 Experimental Setup

We conduct experiments on BlocksWorld benchmark (single-agent) (Valmeekam et al., 2022) and RoCo-benchmark (multi-agent) (Mandi et al., 2023), where agents require multi-step decision-making to achieve the final goal, necessitating the ability to maximize expected future rewards. We compare MLAQ with CoT (Wei et al., 2022), RAP (Hao et al., 2023), REX (Murthy et al., 2023), RAFA (Liu et al., 2023), and RoCo (Mandi et al., 2023). In line with RAP (Hao et al., 2023), we group all tasks by their **optimal steps**, indicating the length of tasks’ optimal trajectory. We also conduct exhaustive additional experiments to verify the effect of the Q-planner, memory, and imagination space.

4.2 Main Results

Single-agent scenario. As shown in Table 1, the optimal steps of BlocksWorld vary from 2 to 12, and the **optimal rate** denotes that the agent reaches target states under optimal steps. Our agent achieves more than 90% optimal rate across all difficulty levels, while the other LLM agents can only succeed in simpler tasks. We also investigate the memory re-utilization ratio in transition imagination. As shown in Figure 3, the ratio of memory re-utilization increases from 19% to 90%, and the average token consumption of each task first increases and then decreases significantly. The increase is attributed to the increased task difficulty, while the decrease results from the increased probability of re-utilization.

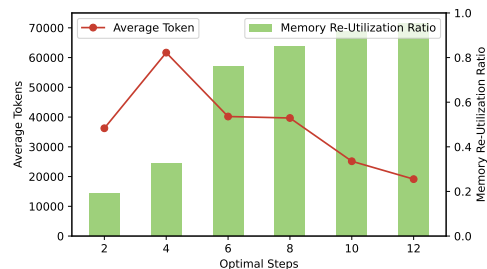


Figure 3: Tendency of memory re-utilization ratio and token consumption.

Multi-agent scenario. Table 2 shows the experimental results of RoCo and MLAQ on Sort domain, where the optimal steps vary from 1 to 6. MLAQ achieves a success rate near 100% across all difficulty levels and maintains an optimal performance exceeding 50% across most tasks, significantly surpassing RoCo especially in difficult tasks. The notation "MLAQ-" denotes that MLAQ agent

Table 2: Comparison of multiple metrics between MLAQ and RoCo in Sort domain.

Metrics	Methods	1-step	2-step	3-step	4-step	5-step	6-step	Average
Success Rate	RoCo	1.00	0.64	0.47	0.10	0.03	0.00	0.35
	MLAQ	1.00	0.96	0.97	1.00	0.93	1.00	0.98
Env Replans (n-shot)	RoCo	0.30	6.30	5.60	9.74	7.67	6.92	6.41
	MLAQ	0.00	0.04	0.03	0.10	0.10	0.04	0.06
Optimal Rate	RoCo	0.80	0.36	0.27	0.00	0.00	0.00	0.21
	MLAQ ⁻	0.95	0.64	0.57	0.67	0.33	0.43	0.58
	MLAQ	1.00	0.86	0.77	0.73	0.50	0.75	0.75
Optimal Gap	RoCo	0.35	3.37	3.43	3.33	2.97	2.00	2.72
	MLAQ ⁻	0.10	0.64	0.73	0.53	1.20	0.46	0.65
	MLAQ	0.00	0.32	0.40	0.43	0.80	0.25	0.39

terminates its imagination upon reaching the target state, without further exploring to get a better policy. The results and analysis on Sandwich domain are provided in Appendix I.3. In Table 2, the **Env Replans** metric denotes the frequency of environment feedback, which reflects the **zero-shot** decision-making ability of the LLM agent. The results indicate that the MLAQ agent almost does not require environmental transitions to correct imaginary transitions. In contrast, existing methods require the environment to verify action’s validity and provide environmental feedback, facilitating LLM agents to replan until actions are available (Mandi et al., 2023; Shinn et al., 2023). We also conduct experiments to compare two types of basic policy (central and dialogue) in Appendix I.1.

4.3 Ablation Results

The key components of MLAQ are the Q-planner, memory, and imagination space. In this section, we validate their effects by individually diminishing their capabilities. The ablation experiments are conducted under 8-step tasks in the BlocksWorld domain. (1) **MLAQ⁻** evaluates the

Table 3: Ablation results in BlocksWorld domain.

Ablation Methods	Optimal Rate	Token	Env Replans
RAP	0.48	-	-
MLAQ	1.00	39688	0.02
w/o domain memory	0.87	378941	0.02
w/o self-examine	0.57	20115	0.81
w/o env-examine	0.95	40772	0.05

impact of the Q-planner’s inability to further optimize. We allow the MLAQ agent to complete only one successful imagination round. The ‘Optimal Gap (OG)’ (gap between completion steps and optimal steps) results in Table 2 shows this deletion of further optimization decreases token consumption while concurrently diminishing the optimal rate. (2) **MLAQ w/o env/self-examine** reduces the quality of imaginary transitions by separately removing self-examine and env-examine. Results show that removing self-examine has a greater impact due to severe hallucination issues, while removing env-examine has little effect as self-examine already improves imaginary data quality well. (3) **MLAQ w/o domain memory** narrows the scope of the memory from domain to task. Results indicate a increase in token consumption alongside a decrease in optimal rate, which stems from the agent’s inability to leverage prior experiences to solve new tasks within the same domain.

5 Conclusion

This work introduces MLAQ, a powerful MDP-based LLM agent framework that achieves zero-shot optimal decision-making. MLAQ consists of a memory module, a Q-planner, and an imagination space to fully leverage the general comprehension capabilities of LLMs and the optimization capabilities of RL. Within the imagination space, the proposed planning algorithm utilizes virtual nodes to perform imaginary interactions, while ensuring a sub-linear regret bound guaranteed by a theorem. Moreover, a mixed-examine mechanism is employed to improve the quality of the imaginary transitions. We evaluate MLAQ on both single-agent BlocksWorld and multi-agent RoCo-benchmark domains, comparing it with several advanced LLM agents. Results indicate that MLAQ outperforms existing methods, especially in tasks involving long-horizon decision-making. The core conclusion of this work is that constructing a model-based RL framework in natural language for current LLMs can further enhance the decision-making abilities of LLM agents.

6 Acknowledgment

This work was supported in part by the Strategic Priority Research Program of Chinese Academy of Sciences under Grant No. XDA27030400, in part by the National Natural Science Foundation of China under Grants 62293541 and 62136008, in part by Beijing Natural Science Foundation under Grant No 4232056, and in part by Beijing Nova Program under Grant 20240484514.

References

- Auer, P., Cesa-Bianchi, N., and Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47:235–256.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- Chai, J., Fu, Y., Zhao, D., and Zhu, Y. (2024). Aligning credit for multi-agent cooperation via model-based counterfactual imagination. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, AAMAS ’24, page 281–289.
- Chen, W., Su, Y., Zuo, J., Yang, C., Yuan, C., Chan, C.-M., Yu, H., Lu, Y., Hung, Y.-H., Qian, C., Qin, Y., Cong, X., Xie, R., Liu, Z., Sun, M., and Zhou, J. (2024). AgentVerse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *The Twelfth International Conference on Learning Representations*.
- Ding, R., Zhang, C., Wang, L., Xu, Y., Ma, M., Zhang, W., Qin, S., Rajmohan, S., Lin, Q., and Zhang, D. (2023). Everything of thoughts: Defying the law of penrose triangle for thought generation. *arXiv preprint arXiv:2311.04254*.
- Egorov, V. and Shpilman, A. (2022). Scalable multi-agent model-based reinforcement learning. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, AAMAS ’22, page 381–390.
- Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2020). Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*.
- Hao, S., Gu, Y., Ma, H., Hong, J., Wang, Z., Wang, D., and Hu, Z. (2023). Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173.
- Hu, G., Zhu, Y., Zhao, D., Zhao, M., and Hao, J. (2023). Event-triggered communication network with limited-bandwidth constraint for multi-agent reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):3966–3978.
- Huang, W., Xia, F., Xiao, T., Chan, H., Liang, J., Florence, P., Zeng, A., Tompson, J., Mordatch, I., Chebotar, Y., Sermanet, P., Jackson, T., Brown, N., Luu, L., Levine, S., Hausman, K., and brian ichter (2022). Inner monologue: Embodied reasoning through planning with language models. In *6th Annual Conference on Robot Learning*.
- Khot, T., Trivedi, H., Finlayson, M., Fu, Y., Richardson, K., Clark, P., and Sabharwal, A. (2022). Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.
- Kocsis, L. and Szepesvári, C. (2006). Bandit based monte-carlo planning. *Machine Learning*, 2006:282–293.
- Kong, Y., Ruan, J., Chen, Y., Zhang, B., Bao, T., shi shiwei, du guo qing, xiaoru hu, Mao, H., Li, Z., Zeng, X., Zhao, R., and Wang, X. (2024). Boosting task planning and tool usage of large language model-based agents in real-world systems. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

- Liu, Z., Hu, H., Zhang, S., Guo, H., Ke, S., Liu, B., and Wang, Z. (2023). Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency. *arXiv preprint arXiv:2309.17382*.
- Mandi, Z., Jain, S., and Song, S. (2023). RoCo: Dialectic multi-robot collaboration with large language models. *arXiv preprint arXiv:2307.04738*.
- Murthy, R., Heinecke, S., Niebles, J. C., Liu, Z., Xue, L., Yao, W., Feng, Y., Chen, Z., Gokul, A., Arpit, D., et al. (2023). REX: Rapid exploration and exploitation for AI agents. *arXiv preprint arXiv:2307.08962*.
- Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., and Whiteson, S. (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 4295–4304.
- Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588(7839):604–609.
- Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning, second edition*. MIT Press.
- Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE.
- Turpin, M., Michael, J., Perez, E., and Bowman, S. (2023). Language models don't always say what they think: Unfaithful explanations in chain-of-thought prompting. In *Advances in Neural Information Processing Systems*, pages 74952–74965.
- Valmeekam, K., Olmo, A., Sreedharan, S., and Kambhampati, S. (2022). Large language models still can't plan (a benchmark for LLMs on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*.
- Valmeekam, K., Stechly, K., and Kambhampati, S. (2024). LLMs still can't plan; can LRMs? a preliminary evaluation of OpenAI's o1 on planbench. *arXiv preprint arXiv:2409.13373*.
- Wang, Z., Cai, S., Chen, G., Liu, A., Ma, X., and Liang, Y. (2023). Describe, explain, plan and select: Interactive planning with LLMs enables open-world multi-task agents. In *Thirty-seventh Conference on Neural Information Processing Systems*.
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Xu, Z., Yu, C., Fang, F., Wang, Y., and Wu, Y. (2023). Language agents with reinforcement learning for strategic play in the Werewolf game. *arXiv preprint arXiv:2310.18940*.
- Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. (2023a). Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822.
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K. R., and Cao, Y. (2023b). ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.
- Zhang, C., Yang, K., Hu, S., Wang, Z., Li, G., Sun, Y., Zhang, C., Zhang, Z., Liu, A., Zhu, S.-C., Chang, X., Zhang, J., Yin, F., Liang, Y., and Yang, Y. (2024a). ProAgent: Building proactive cooperative agents with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17591–17599.

- Zhang, H., Du, W., Shan, J., Zhou, Q., Du, Y., Tenenbaum, J. B., Shu, T., and Gan, C. (2024b). Building cooperative embodied agents modularly with large language models. In *The Twelfth International Conference on Learning Representations*.
- Zhang, Y., Yang, S., Bai, C., Wu, F., Li, X., Li, X., and Wang, Z. (2024c). Towards efficient LLM grounding for embodied multi-agent collaboration. *arXiv preprint arXiv:2405.14314*.
- Zhang, Z., Bo, X., Ma, C., Li, R., Chen, X., Dai, Q., Zhu, J., Dong, Z., and Wen, J.-R. (2024d). A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*.
- Zhao, A., Huang, D., Xu, Q., Lin, M., Liu, Y.-J., and Huang, G. (2024). Expel: LLM agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(17):19632–19642.
- Zhao, Z., Lee, W. S., and Hsu, D. (2023). Large language models as commonsense knowledge for large-scale task planning. In *Advances in Neural Information Processing Systems*, volume 36, pages 31967–31987.
- Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y.-X. (2023a). Language agent tree search unifies reasoning acting and planning in language models. *arXiv preprint arXiv:2310.04406*.
- Zhou, D., Schärli, N., Hou, L., Wei, J., Scales, N., Wang, X., Schuurmans, D., Cui, C., Bousquet, O., Le, Q. V., and Chi, E. H. (2023b). Least-to-most prompting enables complex reasoning in large language models. In *The Eleventh International Conference on Learning Representations*.

A Detailed Descriptions of Memory Scope	10
B Relation to Other Methods	10
C Related Work	10
D Proof for Theorem 1	13
E Limitations	15
F Experimental Environments	16
F.1 BlocksWorld	16
F.2 RoCo-benchmark	16
G Pseudo-code for the Overall Algorithm	19
G.1 Overall algorithm for MLAQ agent	19
G.2 LLM-based Imagination for MLAQ	19
G.3 Expand the replay buffer from the memory	22
G.4 Q function update	23
H Detailed Prompt Template	24
H.1 Basic policy prompt	24
H.2 Action checker prompt	25
H.3 World model prompt	25
H.4 Prediction checker prompt	25
I Detailed Experiment Results	27
I.1 Basic Policy Analysis for Multi-Agent Tasks	27
I.2 Full Results in the Sort domain	27
I.3 Main Results in the Sandwich domain	27
I.4 MLAQ with Multiple Epochs	28
I.5 Self-Examination Analysis	28
J Hyper-Parameters	30
K Prompt Example	31
K.1 Policy Prompt and Response of the State-based Decision-Making	31
K.2 Policy Prompt and Response of the Observation-based Decision-Making	34
K.3 Action Checker Prompt and Response Example	39
K.4 World Model Prompt and Response Example	41
K.5 Prediction Checker Prompt and Response Example	43

A Detailed Descriptions of Memory Scope

MLAQ employs a memory module with domain scope. In order to better illustrate its difference from the memory scope in previous LLM agents, we present the memory module under different scopes in Figure 4. Figure 4 (a) shows a memory with **Step** scope, which only retains the interaction data between the agent and the environment at the current time step. For instance, when refining LLM agent’s decisions, RoCo (Mandi et al., 2023) only relies on the feedback provided by the environment in the current state. Figure 4 (b) shows a memory with **Task** scope, which focuses on solving the current given task and retains the data obtained from the interactions between the agent and the environment in this task. Most LLM agents adopt this form of memory, which is called a **replay buffer**. Take the RAP (Hao et al., 2023) agent as an example, it explores within an LLM-based imagination space and retains visited trajectories to avoid repeated LLM-based state predictions. However, the memory for each task is unique and contains only information relevant to this single task, meaning that the agent cannot gain experience from the decision histories of other tasks within the same domain to solve the current task. Instead, our memory with domain scope stores environmental and imaginary interaction data from all tasks in the domain, effectively reducing the need for LLM queries and improving the efficiency of the reinforcement learning through Q-planner. Besides, one point to **clarify** is that in Figure 4 (c), cylinders are depicted in different colors only to illustrate that they have transitions from different tasks. The transitions stored in our memory do not contain any task-specific labeling, and are always stored in the format of (s, a, r, s') .

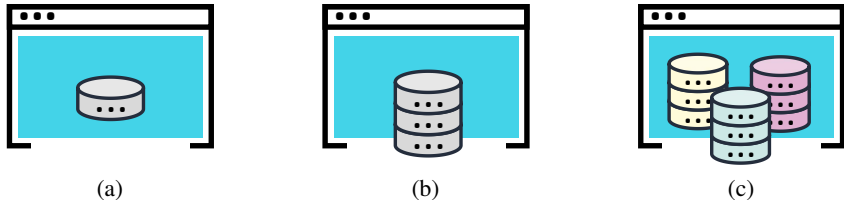


Figure 4: Memory modules with different scope. (a) Step. (b) Task. (c) Domain.

B Relation to Other Methods

In this section, we compare the MLAQ framework with several representative works in LLM agents. Table 4 presents the interaction between Agent, World Model (WM), and Environment (Env) in existing approaches, as well as the forms of their memory. (1) The **Agent-Env** interaction is central to most LLM agents, with efforts to optimize agents through techniques like self-reflect (Shinn et al., 2023) and dialogue (Mandi et al., 2023) for prompt enhancing, and Monte Carlo Tree Search (MCTS) for transition-level optimization (Ding et al., 2023). (2) The **Agent-WM** interaction is originated from model-based RL (Hafner et al., 2020) to expand memory through the agent’s imaginary interactions. RAP (Hao et al., 2023) and ToT (Yao et al., 2023a) construct an LLM-based world model for MCTS-based planning in a single task, but they still not try to narrow the gap between the world model and environment. (3) The **WM-Env** interaction requires using the environmental data to correct the mistakes made by LLM-based world model, which is achieved by LATS (Zhou et al., 2023a). (4) Finally, the memory modules of LLM agents differ in the **level and scope**. Insight-level memory stores agents’ natural language summaries of their insights (Ding et al., 2023), while transition-level memory retains RL-like transitions (Hao et al., 2023). The scope here represents the time span of the data in memories, encompassing (a) Step: the single transition, (b) Task: multiple transitions within the same task, and (c) Domain: multiple transitions in multiple tasks within the same domain. We present different scopes of memory in Appendix A, and detailed related works in Appendix C.

C Related Work

Leveraging the Transformer network architecture, recent advancements in Large Language Models (LLMs) have highlighted their potential in decision-making tasks. These approaches are commonly referred as LLM agents. The research on LLM agents originated from GPT-3, an LLM with 175 billion parameters (Brown et al., 2020). Its authors found that the inference and decision-making

Table 4: Different interactions in existing LLM-based decision-making methods.

Method	Agent-Env	Agent-WM	WM-Env	Memory Level-Scope
Reflexion (Shinn et al., 2023)	Self-reflection	-	-	Insight-Task
Expel (Zhao et al., 2024)	Self-reflection	-	-	Insight-Domain
RoCo (Mandi et al., 2023)	Dialog Feedback	-	-	Transition-Step
XoT (Ding et al., 2023)	MCTS Extraction	-	-	Transition-Task
RAP (Hao et al., 2023)	Direct	MCTS	-	Transition-Task
ToT (Yao et al., 2023a)	Direct	MCTS	-	Transition-Task
LATS (Zhou et al., 2023a)	Self-reflection	MCTS	Env-examine	Transition-Task
MLAQ (Ours)	Q-Planner	UCB-guided	Mixed-examine	Transition-Domain

ability of LLM can be improved by simply adjusting the input prompt without adjusting the model parameters. This insight leads to the development of the Chain-of-Thought (CoT) approach, which improves the LLM’s reasoning capabilities by integrating step-by-step reasoning examples into the prompts (Wei et al., 2022). Subsequent studies have focused on further enhancing LLM reasoning capability through prompt engineering. Techniques such as task decomposition, exemplified by least-to-most (Zhou et al., 2023b) and DecomP (Khot et al., 2022) prompting, are proposed to simplify the task complexity. The aforementioned methods also could address some traditional problems in natural language processing, such as cloze and completion task and reasoning task (Turpin et al., 2023). With the development of LLMs, OpenAI’s O1 has been characterized as a Large Reasoning Model (LRM), demonstrating preliminary planning capabilities (a success rate of 97.8% in BlocksWorld as reported by Valmeekam et al. (2024)). However, due to our experiments employ a more challenging **optimal rate** as the metric, the direct comparison is not presented in the main text.

LLM agents with prompt optimization. As research progresses, an increasing number of studies focus on developing an LLM agent to tackle decision-making tasks with higher complexity. Most methods adopt the existing paradigm, aiming to indirectly enhance the decision-making capabilities of LLM agents by optimizing the prompts given to LLMs (Brown et al., 2020). These approaches typically involve enriching the prompt’s informational content through environmental or human feedback and employing some techniques to summarize these information into some insights, thereby minimizing the prompt’s length (Zhang et al., 2024d). Huang et al. (2022) enable the LLM agent to understand many semantic aspects of the world by leveraging environmental feedbacks to form an inner monologue, acknowledging what, when, and how to do skills in embodied environments. DEPS (Wang et al., 2023) involves the description and self-explanation of the agent’s plans to improve the accuracy of multi-step reasoning in long-horizon tasks. It also provides a trainable goal selector to rank the sub-goals based on the estimated steps of completion, thus refining the agent’s plans. Furthermore, as shown in Table 4, the self-reflect is a widely-used technique to improve LLM agent’s decision-making capability. Reflexion (Shinn et al., 2023) empowers autonomous LLM agents to iteratively revise their actions based on feedback, thereby optimizing their behavior. ReAct (Yao et al., 2023b) introduces a closed-loop reasoning process by alternately generating verbal reasoning traces and actions to achieve self-correction of knowledge. Additionally, it acquires additional information from external sources through the execution of actions. AgentVerse (Chen et al., 2024) involves multiple LLM agents with different roles to orchestrate a collaborative group of expert agents. These agents evaluate their executed actions to adjusting the group and discuss collaboratively for making better decisions. There are also numerous approaches aimed at addressing decision-making problems in multi-agent environments. ProAgent (Zhang et al., 2024a) considers scenarios involving cooperation with human players. It performs cooperative reasoning and planning based on inference of teammates’ behavior, corrects the agents’ beliefs using actual decisions, and stores acquired knowledge into a task memory. Mandi et al. (2023) proposes a multi-agent benchmark RoCo-benchmark with multiple robotic arms solving a cooperative task. It also constructs a dialog-based multi-agent framework RoCo to address the proposed benchmark. CoELA (Zhang et al., 2024b) aims to address the multi-agent cooperation problem with decentralized control by providing communication between LLM agents to exchange their information and knowledge. Its framework extensively utilizes the potential of LLM in perception, communication, planning, memory and execution.

LLM agents with action optimization under MDP framework. Those methods, which only aim at optimizing prompts, cannot achieve satisfactory performance in long-horizon decision-making tasks (Hao et al., 2023), which are more challenging to LLM agents. Firstly, the inherent reasoning

capabilities of these methods, bounded by the fixed parameters of the base LLM, fall short in complex tasks necessitating long-horizon decision sequences. Secondly, adjusting input prompts may improve reasoning to some extent but does not fundamentally optimize the policy of the LLM agent for acquiring **optimal** decision sequences. Recent researches have explored using broader language applications to model the environmental dynamics and reward functions, employing planning algorithms to guide decision-making (Liu et al., 2023; Zhang et al., 2024d). They formulate the LLM agent under the Markov Decision Process (MDP) framework by decomposing original long-horizon decision sequences into atomized state transitions. Yao et al. (2023a) propose Tree-of-Thoughts (ToT), which establishes an incomplete MDP framework limited to state space, action space, and value function components. ToT employs depth-first and breadth-first search techniques to derive optimal sequences of thoughts in the decomposed "thought space". RAP (Hao et al., 2023) leverages LLMs to explicitly construct a world model, serving as the dynamics function within the MDP. RAP employs an LLM-based policy to generate decision trajectories by interacting with the world model within the imagination space and utilizes the MCTS algorithm to effectively balance exploration and exploitation, thus optimizing decision sequences. REX (Murthy et al., 2023) introduces an additional layer of rewards and incorporates principles akin to Upper Confidence Bound (UCB) values to further enrich exploration within the action space, resulting in more robust and efficient agent performance. Everything-of-thought (XoT) (Ding et al., 2023) emphasizes the "Penrose triangle" in traditional LLM agents, indicating that Performance, Efficiency, and Flexibility cannot simultaneously exist. It utilizes LLM for thought generation and employs RL and MCTS algorithms to revise the generated thought tree, thereby enhancing the agent's performance. Zhao et al. (2023) establish an LLM-based commonsense world model and basic policy to conduct MCTS-based searches. External knowledge is obtained through datasets or interactions with human players, enabling updates to the belief of environmental states and consequently making optimal decisions. Kong et al. (2024) integrates an API Retriever, LLM Finetuner, and Demo Selector within a unified framework to tackle the challenges of task planning and tool usage in complex real-world systems. Expel (Zhao et al., 2024) maintains an insight-level memory for the LLM agent. After interacting with the environment, the agent learns from successful trajectories and avoids failed trajectories to summarize a series of insights. These insights are used to enrich the LLM agent's prompts, thereby enhancing its decision-making capabilities. However, most of them require the environmental functions to enhance decision-making and lack the capability to leverage experiences from other tasks within the same domain.

RL approaches in agents. Traditional RL and MARL approaches require RL-based optimization techniques to update the parameters of agent policies. These methods either collect interaction data between agents and the environment (Rashid et al., 2018; Schrittwieser et al., 2020; Hu et al., 2023) or use supervised learning to construct a world model that serves as a digital replica of the real environment (Chai et al., 2024; Egorov and Shpilman, 2022), providing imagined interaction data (Hafner et al., 2020). In contrast, the training of LLMs involves a vast and diverse dataset, enabling the construction of task-specific basic policies and world models through task descriptions in natural language (Hao et al., 2023; Yao et al., 2023a), thereby eliminating the need for parameter updates of traditional methods. Reflexion (Shinn et al., 2023) maintains an RL-like framework, which includes Actor and Critic modules used for action output and action evaluation, respectively. However, it does not employ RL-based optimization at the transition level based on the Bellman equation. Instead, it heavily relies on the decision-making capabilities of the LLM itself, leading to poor performance in long-horizon tasks. Zhang et al. (2024c) utilize RL-based critic regression to learn a sequential advantage function and treat the LLM planner as an optimizer to generate actions that maximize this function. Xu et al. (2023) propose an LLM agent for Werewolf game, which utilizes LLM to generate candidate actions in the current state, and then uses a **one-step** population-based RL training process to select the optimal action from these actions. MCTS is also a traditional RL approach, but it requires prior knowledge about environmental functions and focuses more on the search process. Some of the previously introduced methods (Hao et al., 2023; Zhao et al., 2023) use MCTS for planning and achieve optimal decision-making based on some strong assumptions, but still do not perform satisfactory performance in long-horizon tasks.

D Proof for Theorem 1

Before presenting the proof of Theorem 2, we first introduce Theorem 1 from (Auer et al., 2002) and provide part of the derivation to facilitate the subsequent steps. It is important to note that some variables in this section may cause confusion with similarly defined variables in the main text. These variables are introduced temporarily to improve the readability of the derivation and are only valid within the proof of this section. For instance, while t denotes the time in the main text, it will be used here to denote the number of UCB rounds.

Theorem 1. *For all $m > 1$, if policy UCB is run on state s with m available action, performing each action yields arbitrary Q -value distributions P_1, \dots, P_m with support in $[0, 1]$, then its expected regret after any round number of T is at most:*

$$\left[8 \sum_{i:\Delta_i>0} \left(\frac{\ln N(s)}{\Delta_i}\right)\right] + \left(1 + \frac{\pi^2}{3}\right) \left(\sum_{i=1}^m \Delta_i\right), \quad (4)$$

where $Q(s, 1), \dots, Q(s, m)$ are the expected values of P_1, \dots, P_m .

Proof. The regret is defined as follows:

$$\text{Regret} = \sum_{j:\Delta_j>0} \Delta_j \mathbb{E}[C_j(T)], \quad (5)$$

where $\Delta_i = Q^* - Q(s, i)$, $Q^* \doteq \max_i \{Q(s, i)\}$, and $C_i(T)$ denotes the selection number of machine i over the first T rounds. The key of the derivation lies in the decomposition of $\mathbb{E}[C_i(T)]$. Due to the initial evaluation conducted by UCB for each machine, $C_i(T)$ can be rearranged as:

$$C_i(T) = \sum_{t=1}^T \{I_t = i\} = 1 + \sum_{t=m+1}^T \{I_t = i\}, \quad (6)$$

where $\{\cdot\}$ is an indicator function. If the event $\{I_t = i\}$ occurs, meaning that the UCB policy selects machine i in round t , then the indicator function outputs 1. Otherwise, it outputs 0.

Auer et al. (2002) derived the conclusion of the theorem through a step-wise approach using this equation. For a detailed derivation process, please refer to the original proof in (Auer et al., 2002). The subsequent derivations in this paper require only the few steps outlined above. \square

Theorem 2. *Suppose the player employs the node selection rule defined in (2). Then, in a scenario with m available actions in state s , the bound of the expected cumulative regret is:*

$$\text{Regret} = \sum_{j:\Delta_j>0} \Delta_j \mathbb{E}[C_j(N(s))] \leq \left[8 \sum_{i:\Delta_i>0} \left(\frac{\ln N(s)}{\Delta_i}\right)\right] + \left(1 + \frac{\pi^2}{3}\right) \left(\sum_{i=1}^m \Delta_i\right), \quad (7)$$

where $\Delta_i = Q^* - Q(s, i)$ for action index i in $\{1, 2, \dots, m\}$ and $Q^* \doteq \max_i \{Q(s, i)\}$, $N(s)$ is the visit count of s , and $C_i(N(s))$ is the selection number of action i .

Proof. Since the definition of regret has not changed, we will continue our derivation starting from the decomposition of $C_i(T)$, where T has the same meaning with $N(s)$. We will ultimately demonstrate that, due to the presence of the virtual node y , the player can achieve the same upper bound on regret as UCB. In the multi-armed bandit variant, action i can be selected only if y has been selected at least i times. Thus, we define the selection number of y at round t as $A(t)$, and we have:

$$\begin{aligned} C_i(T) &= \sum_{t=1}^T \{I_t = i, A(t) \geq i\} = \sum_{t=i+1}^T \{I_t = i, A(t) \geq i\} \\ &= \sum_{t=i+1}^T \{I_t = i, A(t-1) \geq i\} \cup \{I_t = i, \xi(t-1, i)\} \\ &= \sum_{t=i+2}^T \{I_t = i, A(t-1) \geq i\} + \sum_{t=i+1}^T \{I_t = i, \xi(t-1, i)\}, \end{aligned} \quad (8)$$

where $\xi(t, i)$ denotes the event in which player select y for the i -th time at round t . The derivation in the first line follows from the fact that $A(1) = 0$ and $A(t) \leq t - 1$. Therefore, when $t \leq i$, the event $A(t) \geq i$ cannot occur. The derivation from the first line to the second line primarily relies on rewriting the original event $A(t) \geq i$ as the union of two mutually exclusive events, $A(t - 1) \geq i$ and $\xi(t - 1, i)$. By repeatedly applying the above decomposition operation to the first term of the formula, we have:

$$\begin{aligned} C_i(T) &= \sum_{k=1}^{n-i} \sum_{t=i+k}^n \{I_t = i, \xi(t - k, i)\} = \sum_{t=i+1}^n \sum_{k=1}^{t-i} \{I_t = i, \xi(t - k, i)\} \\ &= \sum_{t=i+1}^n \sum_{x=i}^{t-1} \{I_t = i, \xi(x, i)\} = \sum_{x=i}^{n-1} \sum_{t=x+1}^n \{I_t = i, \xi(x, i)\} \end{aligned} \quad (9)$$

By setting $x = t - k$ and exchanging a series of inner and outer loops, we obtain a more interpretable formula. This formula first determines the round in which y is selected for the i -th time by iterating the variable x over the interval $[i, n - 1]$, followed by iterating the time t to compute the selection number of action i . Clearly, the above expression is less than n , leading directly to a linear regret bound. However, to achieve the goal of deriving a sub-linear bound, we should use the following fact:

$$\begin{aligned} C_i(T) &\leq \max_x \sum_{t=x+1}^n \{I_t = i, \xi(x, i)\} = \max_x \sum_{t=x+1, n} \{I_t = i\} \\ &\leq \sum_{t=1}^n \{I_t = i\} \end{aligned} \quad (10)$$

Thus, we can directly proceed to Equation (6) and its subsequent derivations, ultimately proving that in this case, the player can still achieve the same regret bound:

$$\left[8 \sum_{i: \Delta_i > 0} \left(\frac{\ln N(s)}{\Delta_i} \right) \right] + \left(1 + \frac{\pi^2}{3} \right) \left(\sum_{i=1}^m \Delta_i \right), \quad (11)$$

where $N(s)$ is equivalent to T . □

In the calculation of the vUCB value in (3), the confidence bound for virtual nodes includes an additional coefficient $g(x)$ compared to that of actual nodes. Assuming the number of available actions is m , we expect the virtual node to be selected at most m times, and the coefficient should decrease as the size of $\hat{\delta}(s)$ increases. Therefore, it should possess the following characteristics:

$$g(m) = 0 \text{ and } \frac{\partial g}{\partial x} \leq 0 \quad (12)$$

However, since the number of available actions cannot be known in advance when planning in the LLM-based imagination space without environmental tools, the first property cannot be satisfied. Therefore, in practice, we use $g(x) = w_g(e^{-x^2} - \epsilon_g)$ as a substitute, whose value rapidly diminishes as the size of $\hat{\delta}(s)$ increases.

E Limitations

The first limitation of this work is that the MLAQ agent requires the environment to provide a brief description of the domain in natural language, which needs to be manually written and should provide as much information as possible to help LLM understand this domain.

The second limitation of this work is the accuracy of our self-examine mechanism, especially the action checker. In order to minimize the reliance of the LLM agent on prior knowledge of the environment, we implement an LLM-based self-examine mechanism using an LLM-based action checker and prediction checker. Although the MLAQ agent with this self-examine mechanism could effectively improve its fault tolerance as demonstrated in Appendix I.5, the checkers cannot always perfectly identify errors. Consequently, the incorrect transitions may be added to the memory, affecting the optimality of the agent's decisions. For some practical application problems, users do not need to hold the MLAQ's generality. By providing it with ground-truth environmental action and state checkers, the generality is decreased but the agent's performance in practical scenarios could be further enhanced.

F Experimental Environments

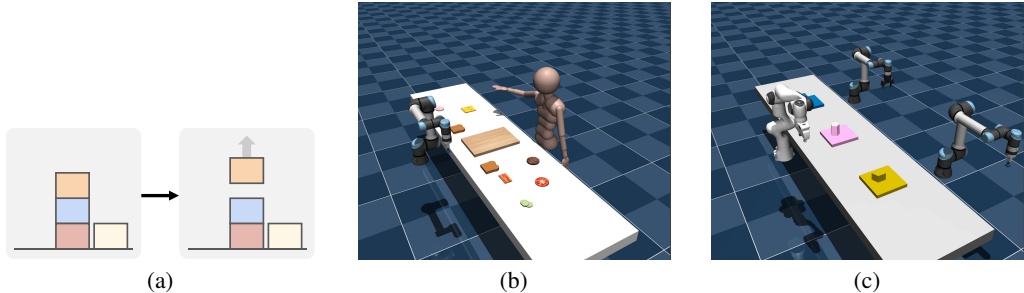


Figure 5: Experimental environments. (a) BlocksWorld. (b) Sandwich domain of RoCo-benchmark. (c) Sort domain of RoCo-benchmark.

As shown in Figure 5, we evaluate the methods in BlocksWorld (single-agent) and RoCo (multi-agent) benchmarks. Next, we will provide a detailed description of each task, as well as the state and action space.

F.1 BlocksWorld

BlocksWorld is a well-known single-agent benchmark for evaluating LLM agents (Valmeekam et al., 2022), where an agent should rearrange some blocks into multiple stacks in a particular order. In this work, we choose the tasks with four blocks to evaluate our methods. Hao et al. (2023) has grouped all tasks according to the optimal step, and we randomly choose up to 30 tasks from each group for evaluation. As shown in Figure 6 (a), the state space in the original BlocksWorld’s codes has a lot of redundant information, which may help agents make better decisions, but may lead to errors when making predictions by the world model. Therefore, as shown in Figure 6 (b), we adjust the format of them in a more concise and clear manner without affecting the validity of states.

The **state** records what the agent’s hand is holding: (‘Empty’ or ‘Holding <block>’) and what each block is on (‘on <block>’, ‘on table’, or ‘in hand’). The **action** is an instruction that moves one block, which is one of STACK, UNSTACK, PUT, and PICK UP.

1. PICK UP <object>: Execute if hand is empty, and <object> is on table and no block is on it. Pick up <object> from table. After execution, the agent will be holding the <object>.
2. UNSTACK <object>: Execute if hand is empty, <object> is on another block and no block is on it. UNSTACK <object> from another block. After execution, the agent will be holding the <object>.
3. PUT DOWN <object>: Execute if hand is holding <object>. Put down <object> on table. After execution, the agent’s hand will be empty.
4. STACK <object> ON <target>: Execute if hand is holding <object>, and no block is on <target>. Stack <object> on the top of <target>. After execution, the agent’s hand will be empty.

F.2 RoCo-benchmark

RoCo-benchmark is built upon the Mujoco engine (Todorov et al., 2012) to construct several multi-arm scenarios. RoCo-benchmark provides 6 different tasks, including those containing two agents and those containing three agents, and the robotic arms in them have their own movable range and require full cooperation to complete the given task. We choose two of these tasks, Sort and Sandwich, to evaluate our method. In Roco-benchmark’s codes, there is no fixed format for the state space, and the states are intermingled in the form of information within the agent’s prompt. Therefore, we have constructed the state spaces for these two tasks in a concise manner, while keeping the action spaces consistent with the original codes.

Sandwich domain. There are two robots, Chad and Dave, trying to make a sandwich following the given recipe order. Chad can only reach for food items on the right side of the table, while

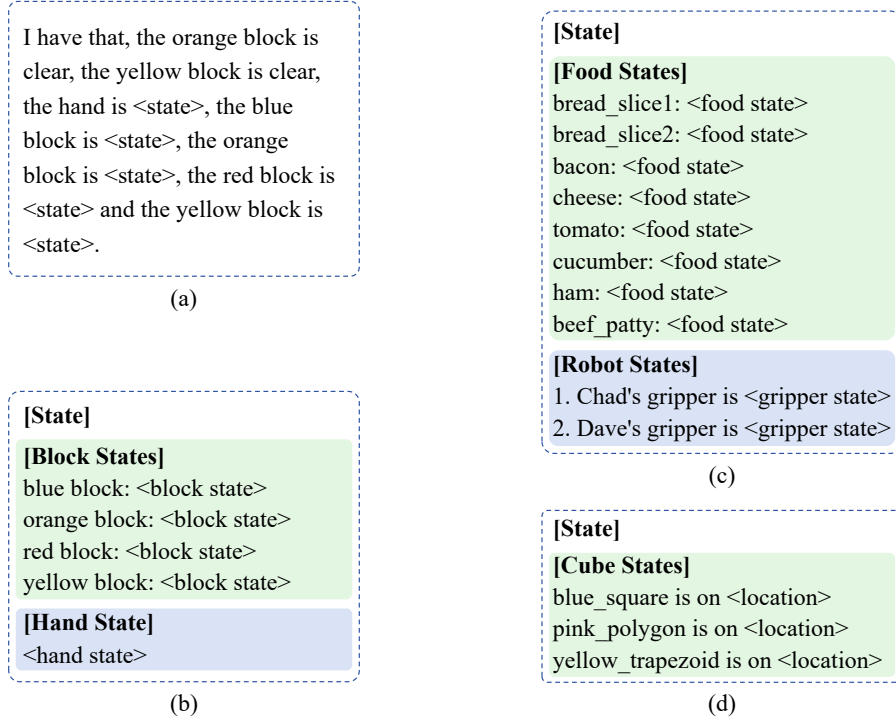


Figure 6: State templates. (a) Template in original BlocksWorld codes. (b) Modified Template in BlocksWorld. (c) Template in Sandwich domain of RoCo-benchmark. (d) Template in Sort domain of RoCo-benchmark.

Dave can only reach for food items on the left side, with each food item starting in a fixed position. The robots need to place the food items on the cutting board in the middle of the table one by one, following the order given in a recipe. Different types of sandwiches contain different combinations of food items. Take the ‘bacon’ sandwich as an example, it requires ‘bread_slice1’, ‘bacon’, ‘cheese’, ‘tomato’, and ‘bread_slice2’. In a bacon recipe, apart from the fixed positions of ‘bread_slice1’ and ‘bread_slice2’ as the two ends of the sandwich, the order of the other food items can vary across different recipes. Therefore, one type of sandwich may have multiple different recipes. We group all the recipes according to the optimal step to evaluate the agents. In the decision-making process, robots cannot execute PUT actions simultaneously within one timestep to avoid collisions.

As shown in Figure 6 (c), the **state** records the state of each food item (‘on left/right side’, ‘atop <another food item> or <cutting_board>’, or ‘gripped by <robot>’) and the state of each robot’s gripper (‘holding <food item>’ or ‘empty’). The **action** is an instruction for each agent to interact with the food items, which is one of PICK, PUT, and WAIT.

1. PICK <obj>: Execute if gripper is empty and the food item’s state is not ‘atop <obj>’. Pick the <obj> from the table. After execution, the robot’s gripper will be holding <obj>.
2. PUT <obj1> <obj2>: Execute if gripper is holding <obj1>. Put <obj1> on the top of <obj2>. After execution, the robot’s gripper will be empty.
3. WAIT: Do nothing.

The action checker in the Sandwich environment strictly limiting the placement of food items to follow the recipe, rather than allowing it to be placed in an incorrect order. Therefore, the size of agents’ available action set in each state is relatively small. The agent can only choose to wait, pick up the next food item, or place the food item from the gripper according to the recipe, so that RoCo could achieve a high success rate in this task, but the optimal rate is still low.

Sort domain. There are three robots, Alice, Bob, and Chad, trying to sort three cubes onto their corresponding panels. There are a total of seven panels. Alice can access panels 1 to 3, Bob can access panels 3 to 5, and Chad can access panels 5 to 7. The three cubes are a blue square, a pink

polygon, and a yellow trapezoid, with their respective target panels being panel2, panel4, and panel6. In the decision-making process, a panel cannot contain two or more cubes, and a cube cannot be grasped by two robots simultaneously.

As shown in Figure 6 (d), the **state** records the state of three cubes (on <panel>) and the action is an instruction for each agent to interact with the cubes, which is one of PICK & PLACE and WAIT.

1. PICK <object> PLACE <target>: Pick up <object> and place it onto <target>, where <object> is a cube and <target> is a panel.
2. WAIT: Do nothing.

Given the complexity of this domain involving more agents and expansive available action spaces, its difficulty surpasses that of the Sandwich domain. In the Sandwich domain, decision trajectories are linear and decision loops are non-existent, meaning that the multi-agent system cannot revert to its initial state after a series of decisions. However, in the Sort domain, the occurrence of loops adds to the challenge of reaching target states. MLAQ could address this challenge by leveraging RL-based optimization to make long-term optimal decisions.

G Pseudo-code for the Overall Algorithm

We present detailed pseudo-code for our MLAQ framework in this section, and these algorithms are used to fully describe the overall process presented in Figure 1 and Figure 2. We have simplified the entire algorithm process into the flowchart in Figure 7 to facilitate a better understanding of how MLAQ makes optimal decisions for new a task.

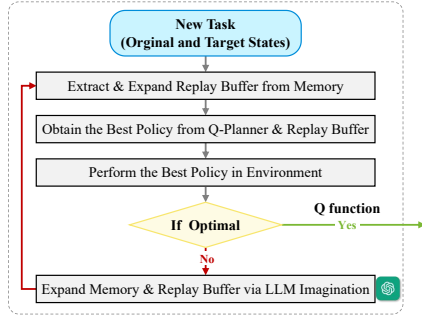


Figure 7: Flowchart of MLAQ agent to obtain the optimal decision sequence for a new task, with the input being the domain description and domain-specific memory and the output being the Q function.

G.1 Overall algorithm for MLAQ agent

At first, Algorithm 1 shows how MLAQ agent obtains a optimal decision sequence in a given task. Its inputs are the environment dynamics and reward function with the domain description in natural language τ to provide interaction between the true environment and the agent (or multi-agent system). At the beginning of the optimization of MLAQ agent, we initialize an empty memory \mathcal{M} and a Q function $Q(\cdot, \cdot)$ to set all Q values to -1. As described in the experiment section, we then scan all tasks of a domain to gradually expand the memory module through environmental and imaginary interactions.

For each single task, we have a unique pair of original state s_0 and target state \bar{s} , and initialize a replay buffer \mathcal{D} for this task. Before making decisions in this task, MLAQ agent will try to find this task’s best trajectory from the memory to expand the empty replay buffer, and a task-specific Q function $Q(\cdot, \cdot)$ will be obtained based on \mathcal{D} . If the agent fails in finding an available trajectory from the memory, meaning that the $Q(s_0, a)$ is no larger than the threshold \bar{Q} for any action a , we would skip the following environmental interaction phase to directly perform the imagination process until an available decision sequence is found. Otherwise, the agent interacts with the environment by making best decisions according to $Q(\cdot, \cdot)$ to evaluate the optimality $\mathbb{1}(s_0, \bar{s})$ of the resulting decision sequence. The indicator $\mathbb{1}(s_0, \bar{s})$ is set to True only if the environmental sequence is optimal.

In order to expand the memory and replay buffer to optimize the agent, we keep performing imaginary interactions until the optimal sequence is obtained for the indicator $\mathbb{1}(s_0, \bar{s})$ is True or the maximum trial number K_c is reached. As more tasks are scanned, the size of memory \mathcal{M} will gradually increase, and we will finally output it to assist the MLAQ agent in making optimal decisions under any tasks in this domain. While for the tasks have not been scanned before, MLAQ would also use the transitions stored in \mathcal{M} to reduce LLM queries and obtain the optimal decision sequence much easier.

G.2 LLM-based Imagination for MLAQ

Algorithm 2 input the original state s_0 and target state \bar{s} with domain description τ . The memory \mathcal{M} contains transitions from other tasks and the temporary replay buffer \mathcal{D} is specific to the current task. At the beginning, we should first add a virtual node (if does not exist) for the original state s_0 to start the imaginary interactions. Then, we perform the phases of node selection and transition imagination to expand the memory and replay buffer, and the overall process has already described in section 3. The outputs of the imaginary interaction are the expanded memory and replay buffer. It should be noted that after each step of transition imagination, we would perform an "Expand-Buffer" method to find an available trajectory with better optimality within \mathcal{M} and add nodes along this trajectory to \mathcal{D} .

Algorithm 1: Obtaining optimal decision sequence for MLAQ agent

```
1 Inputs:
2   The environment, including dynamics  $s' = \mathbb{T}(s, a)$  and reward function  $r = \mathbb{R}(s, a, s')$ ;
3   Task description in natural language  $\tau$ ;
4 Initialize:
5   memory  $\mathcal{M}$  and Q function  $Q(\cdot, \cdot) = -1$ ;
6 for task = 1, 2, 3, ... do
7   # Each task has a unique original state  $s_0$  and target state  $\bar{s}$ 
8   Initialize a replay buffer  $\mathcal{D}$  for this task;
9   Set the optimal sequence indicator for this task  $\mathbb{1}(s_0, \bar{s})$  to be False;
10  while  $\mathbb{1}(s_0, \bar{s})$  is not True AND trial <  $K_c$  do
11    # Expand the replay buffer and get a task-specific Q function based on the memory
12     $\mathcal{D} = \text{Expand-Buffer}(s_0, \bar{s}; \mathcal{M}, \mathcal{D})$ ;
13     $Q(\cdot, \cdot) = \text{Q-Update}(s_0, \bar{s}; \mathcal{D})$ ;
14    if  $\max_a Q(s_0, a) < \bar{Q}$  then
15      # An available trajectory does not exist based on  $\mathcal{D}$ 
16      Skip the environmental interaction phase and turn to the next trial of imagination;
17    end
18    # Environmental Interaction Phase
19    while  $s_t$  is not  $\bar{s}$  AND  $t < T$  do
20      Get the optimal action  $a_t = \arg \max_a Q(s_t, a)$ ;
21      if  $a_t$  is available for the environment then
22        Perform action to the environment  $s_{t+1} = \mathbb{T}(s_t, a_t)$  and  $r_t = \mathbb{R}(s_t, a_t, s_{t+1})$ ;
23      else
24        Remove transition  $(s_t, a_t, r(s_t, a_t), c(s_t, a_t))$  from  $\mathcal{M}$  and  $\mathcal{D}$ ;
25        Get environmental feedback  $\mathcal{F}$  and store  $(s_t, a_t, \mathcal{F})$  into  $\mathcal{M}$  and  $\mathcal{D}$ ;
26      end
27      Store transition  $(s_t, a_t, r_t, s_{t+1})$  into  $\mathcal{M}$  and correct the wrong transition;
28       $s_t \leftarrow s_{t+1}$ ;
29    end
30    if  $s_t$  is  $\bar{s}$  then
31       $\mathbb{1}(s_0, \bar{s})$  is True if length of decision sequence is optimal else False;
32    else
33       $\mathbb{1}(s_0, \bar{s})$  is False;
34    end
35    if  $\mathbb{1}(s_0, \bar{s})$  is not True then
36      # Perform a round of imaginary interaction to expand the replay buffer and memory
37       $\mathcal{M}, \mathcal{D} = \text{Imagination}(s_0, \bar{s}; \tau, \mathcal{M}, \mathcal{D})$ ;
38    end
39  end
40 end
41 Outputs:
42   memory  $\mathcal{M}$ ;
```

Algorithm 2: Imagination($s_0, \bar{s}; \tau, \mathcal{M}, \mathcal{D}$) guided by UCB values

```
1 Inputs:
2   Original state  $s_0$  and target state  $\bar{s}$  of this task with domain description in natural language  $\tau$ ;
3   The memory  $\mathcal{M}$  and replay buffer  $\mathcal{D}$ ;
4 Initialize:
5   Add a virtual node as the first child node of the original state  $s_0$ , and  $N(s_0) \leftarrow N(s_0) + 1$ ;
6 while  $s_t$  is not  $\bar{s}$  AND  $t < T$  do
7   # Node Selection Phase
8   Get the optimal action  $a_t = \arg \max_a v\text{UCB}(s_t, a, c(s_t, a))$ ;
9   if  $c(s_t, a_t)$  is a virtual node then
10    # Transition Imagination Phase
11    Action available indicator  $\hat{\mathbb{1}}(a_t) \leftarrow \text{False}$ ;
12    # Basic policy makes decisions and action checker verifies if the output is available
13    while  $\mathbb{1}(a_t)$  is not True AND trial  $< K_a$  do
14      Get a non-virtual action through LLM-based basic policy  $a_t \sim \pi(s_t; \tau, \mathcal{M})$ ;
15      if  $(s_t, a_t)$  in memory  $\mathcal{M}$  then
16        Retrieve transition data  $(s_t, a_t, r(s_t, a_t), c(s_t, a_t))$  from  $\mathcal{M}$  to  $\mathcal{D}$ ;
17         $\hat{\mathbb{1}}(a_t) = \text{True}$ ;
18      else
19        Check the action  $\hat{\mathbb{1}}(a_t) = \xi_a(s_t, a_t; \tau, \mathcal{M})$ ;
20      end
21    end
22    ▷ This trajectory is failed and terminated if  $\mathbb{1}(a_t)$  is False;
23    if  $\mathbb{1}(a_t)$  is True then
24      # World model predicts and prediction checker verifies if the output is available
25      Prediction available indicator  $\hat{\mathbb{1}}(s_{t+1}) \leftarrow \text{False}$ ;
26      while  $\hat{\mathbb{1}}(s_{t+1})$  is not True AND trial  $< K_s$  do
27        Predict the next state  $s_{t+1} \sim \hat{\mathbb{T}}(s_t, a_t; \tau, \mathcal{M})$ ;
28        Check the prediction  $\hat{\mathbb{1}}(s_{t+1}) = \xi_s(s_t, a_t, s_{t+1}; \tau, \mathcal{M})$ ;
29      end
30    end
31    ▷ This trajectory is failed and terminated if  $\mathbb{1}(s_{t+1})$  is False;
32  else
33     $s_{t+1} = c(s_t, a_t)$ ;
34    if  $s_{t+1}$  is not in  $\mathcal{D}$  then
35      Add a virtual node as the first child node of  $s_{t+1}$ ;
36    end
37  end
38   $s_t \leftarrow s_{t+1}$  and  $N(s_t) \leftarrow N(s_t) + 1$ ;
39  if  $s_t$  is  $\bar{s}$  then
40     $r_t = +1$ ;
41  else if  $s_t$  is a failure state then
42     $r_t = -1$ ;
43  else
44     $r_t = 0$ ;
45  end
46  # Expand the replay buffer and memory
47  Store transition  $(s_t, a_t, r_t, s_{t+1})$  into the replay buffer and memory (if does not exist);
48  # Expand the replay buffer from memory if possible
49   $\mathcal{D} = \text{Expand-Buffer}(s_0, \bar{s}; \mathcal{M}, \mathcal{D})$ ;
50 end
51 Outputs:
52 The expanded memory  $\mathcal{M}$  and replay buffer  $\mathcal{D}$ ;
```

G.3 Expand the replay buffer from the memory

Algorithm 3 is used to expand the task-specific replay buffer based on the transitions in the memory. It firstly finds an available trajectory with better optimality from \mathcal{M} and then store the transitions of this trajectory in the replay buffer \mathcal{D} . This will effectively improve the exploration efficiency of the MLAQ agent within the imagination space. For example, if the replay buffer is empty and there exists an available trajectory from s_0 to \bar{s} , then the nodes and edges in this trajectory will be added to \mathcal{D} . Therefore, the first trial in Algorithm 1 would not require any LLM queries to get an available policy for this task.

Furthermore, if \mathcal{D} is not empty, this algorithm could also help the MLAQ agent to aggregate transitions from memory for enhancing optimal decision-making capability in complex tasks. For example, if the agent explores a new transition and the next state corresponding to this transition has a trajectory with better optimality towards the target state \bar{s} , the MLAQ agent could add the nodes in this subsequent trajectory to \mathcal{D} without extra exploration within the imagination space.

Algorithm 3: Expand-Buffer($s_0, \bar{s}; \mathcal{M}, \mathcal{D}$)

```
1 Inputs:  
2   Original state  $s_0$  and target state  $\bar{s}$  of this task with task description in natural language  $\tau$ ;  
3   The memory  $\mathcal{M}$ ;  
4    $Q(\cdot, \cdot) = \text{Q-Update}(s_0, \bar{s}; \mathcal{M})$ ;  
5 if  $\max_a Q(s_0, a) > \tilde{Q}$  then  
6    $s_t \leftarrow s_0$ ;  
7   while  $s_t$  is not  $\bar{s}$  do  
8     Get the optimal action  $a_t = \arg \max_a Q(s_t, a)$ ;  
9     Retrieve transition data  $(s_t, a_t, r(s_t, a_t), c(s_t, a_t))$  from  $\mathcal{M}$  to  $\mathcal{D}$  (if does not exist);  
10     $s_t \leftarrow c(s_t, a_t)$ ;  
11  end  
12 end  
13 Outputs:  
14  The expanded replay buffer  $\mathcal{D}$ ;
```

G.4 Q function update

Algorithm 4 is employed to get a Q function for the given task s_0 and \bar{s} on the basis of \mathcal{X} , where \mathcal{X} could be the memory \mathcal{M} and the replay buffer \mathcal{D} . At first, we initialize the Q function to set all Q values to -1, and assign rewards for all transitions based the target state \bar{s} . Then, the algorithm would perform multiple loops of Q-learning process to get the approximated Q function $Q(\cdot, \cdot)$. This algorithm will be used by the Q-Planner in Algorithm 1 and the replay buffer expansion process in Algorithm 3.

Algorithm 4: Q-Update($s_0, \bar{s}; \mathcal{X}$)

```
1 Inputs:  
2   The original state  $s_0$  and target state  $\bar{s}$  of the given task;  
3   The memory module or replay buffer  $\mathcal{X}$ ;  
4 Initialize:  
5   Initialize a Q function  $Q(\cdot, \cdot) = -1$ ;  
6   For all transition  $(s, a, r, s')$  in  $\mathcal{X}$ , if  $s'$  is  $\bar{s}$ , then  $r = +1$ ; if  $s'$  is a failure state, then  $r = -1$ ;  
   otherwise,  $r = 0$ ;  
7 for loop  $k = 1, 2, 3, \dots$  do  
8   for  $(s_t, a_t, r_t, s_{t+1}) \in \mathcal{X}$  do  
9     Perform the following Q-learning update rule for this transition;  
10    
$$Q_{k+1}(s_t, a_t) = Q_k(s_t, a_t) + \alpha \left( r_t + \gamma \max_{a'} Q_k(s_{t+1}, a') - Q_k(s_t, a_t) \right)$$
  
11   end  
12 end  
13 Outputs:  
14   The Q function  $Q(\cdot, \cdot)$ ;
```

H Detailed Prompt Template

This section presents the detailed prompt templates for the basic policy, world model, action checker, and prediction checker. The prompt template has multiple parts to provide different types of information for the modules. The prompts in the purple cube are the component-specific prompts to guide the agent in making decisions, predicting next states, checking actions or predictions. In the prompt, we guide the corresponding modules step-by-step to enhance the accuracy of their outputs. The prompts in the green cube contain some task-specific information, and prompts in the blue cube contain some self-provided information generated from self-examinations and vUCB-based planning. The prompts in grey cube is the ‘user prompt’, which contains the current information for the modules.

H.1 Basic policy prompt

The prompt shown in Figure 8 requires the current state and target state in natural language to make decisions. In the **[Detailed Instruction for Policy]** part, we decompose the decision-making process into five parts: [Action Planning], [Action Conclusion], [Action Constraints Check], [Forbidden Action Check], and [Action Output], gradually guiding the agent to make decisions that satisfy the constraints. As for the basic policy of multi-agent systems, we directly follow the prompt design of RoCo, and the prompts can be found in (Mandi et al., 2023). The policy mistakes are the environmental feedbacks of previous wrong actions stored in the memory \mathcal{M} . The policy feedbacks are the temporary feedbacks provided by the self-examination only under the current state. The forbidden actions are the non-virtual actions of the current state node’s child nodes.

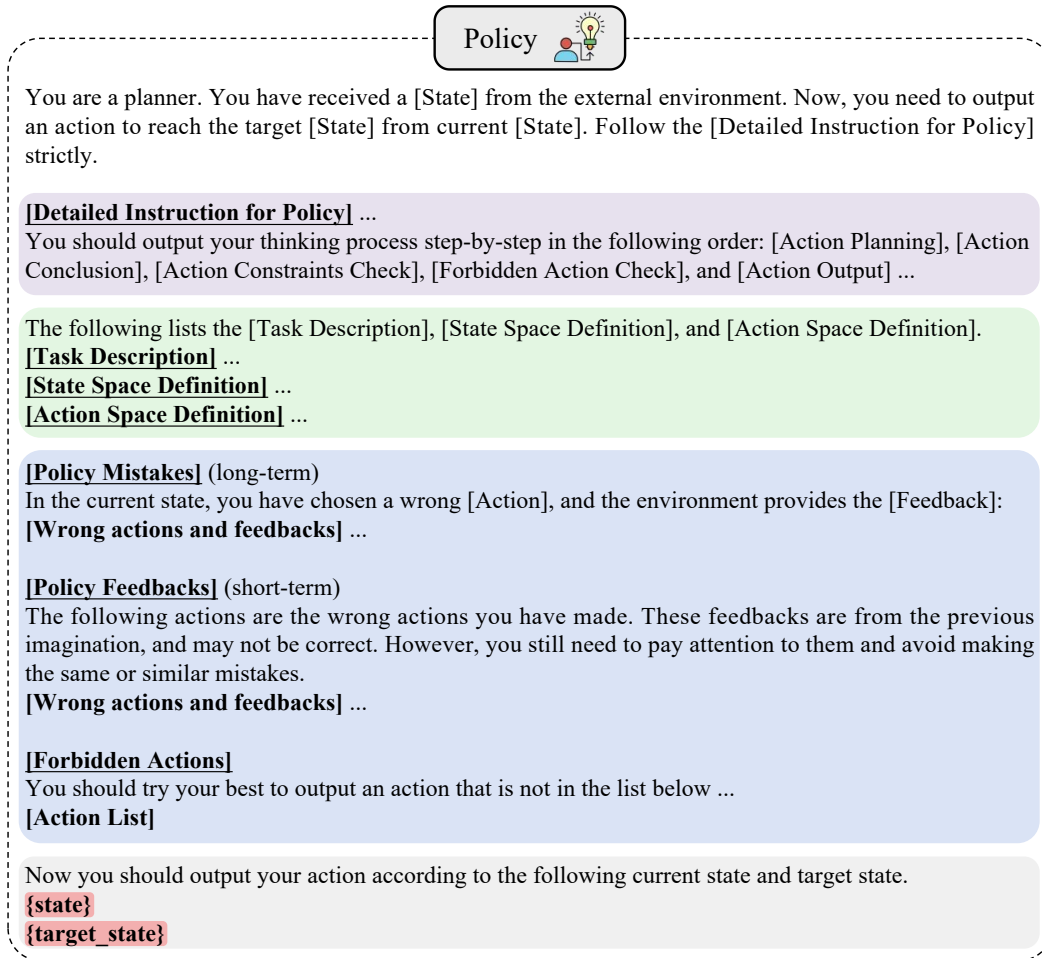


Figure 8: A simplified template for the basic policy prompt.

H.2 Action checker prompt

The prompt shown in Figure 9 requires the current state and the action should be checked. In the **[Checker Instruction]** part, we decompose the checking process into three parts: [Action Constraints Check], [Conclusion Justification], and [Checker Conclusion]. We provide the previous mistakes and the corresponding feedbacks stored in the memory under this current state for the action checker to avoid the same mistakes.

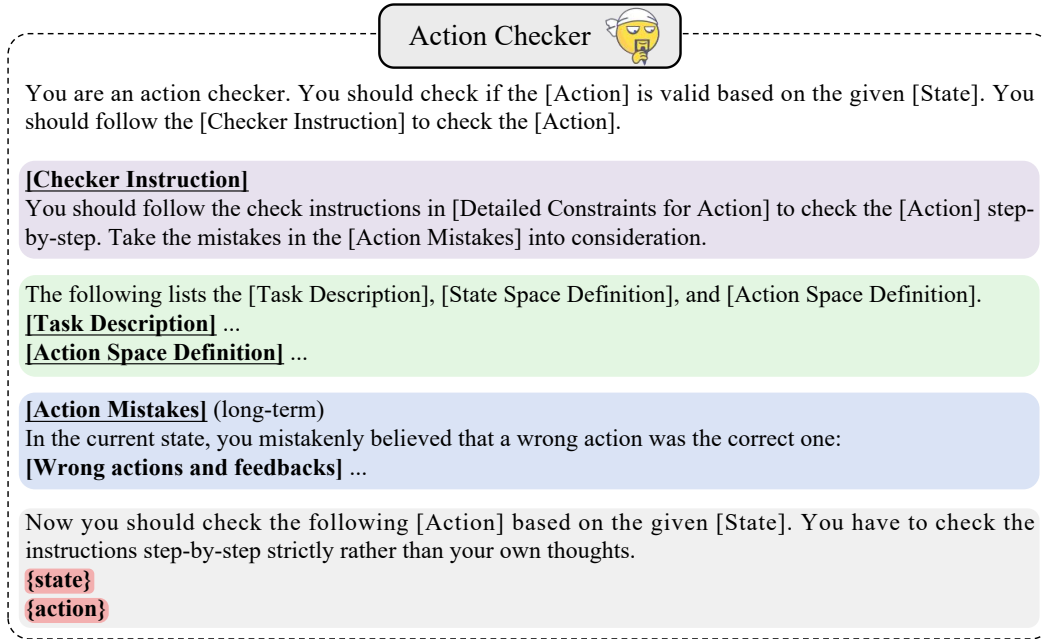


Figure 9: A simplified template for the action checker prompt.

H.3 World model prompt

The prompt shown in Figure 10 requires the current state and action to predict the next state. In the **[Predicting Instruction]** part, we decompose the predicting process into three parts: [Interaction Item Pool], [Action Forward Rule] and [Prediction Conclusion]. The prompt also contains the feedbacks from the long-term and replay buffer. In fact, LLM can better handle the task of predicting with step-by-step reasoning, and we also find the prediction accuracy of the world model to be nearly 100% in the experimental section.

H.4 Prediction checker prompt

The prompt shown in Figure 11 requires the current state, executed action, and the prediction should be checked. In the **[Checker Instruction]** part, we decompose the checking process into three parts: [Prediction Format Check], [Prediction Rule Check], and [Checker Conclusion]. We also provide the previous mistakes and the corresponding feedbacks stored in the memory under this state-action pair for the prediction checker to avoid the same mistakes.

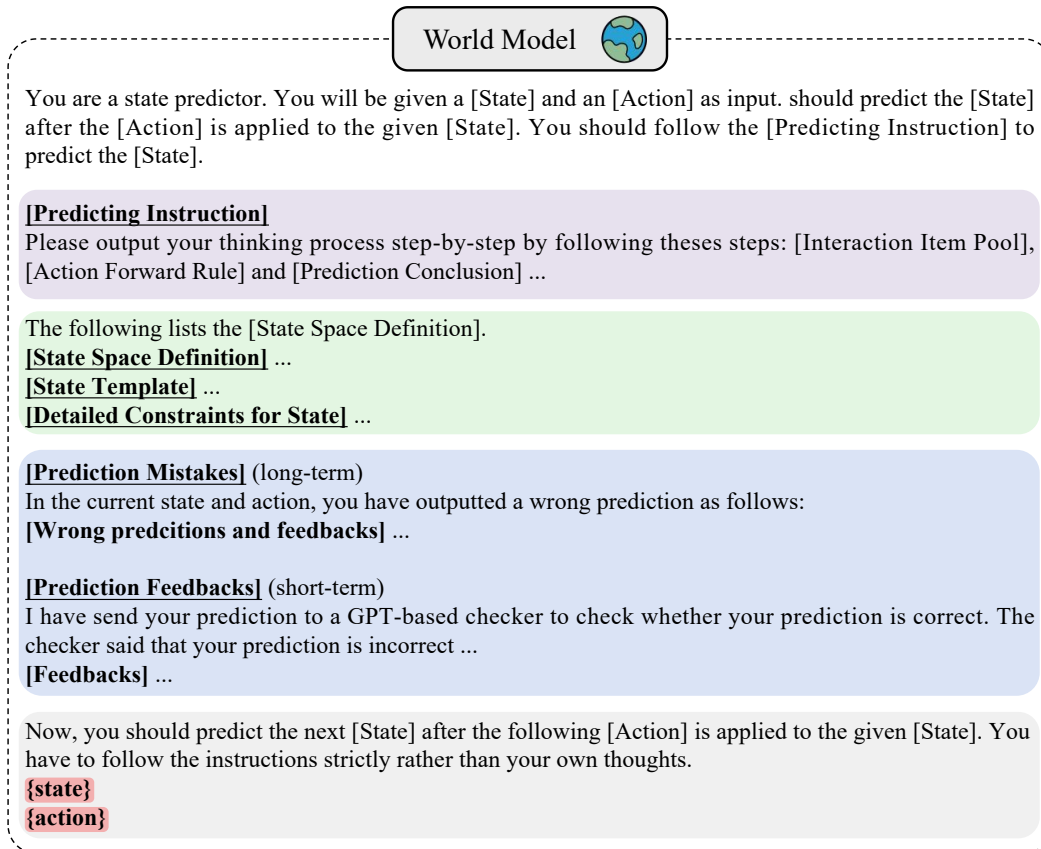


Figure 10: A simplified template for the world model prompt.

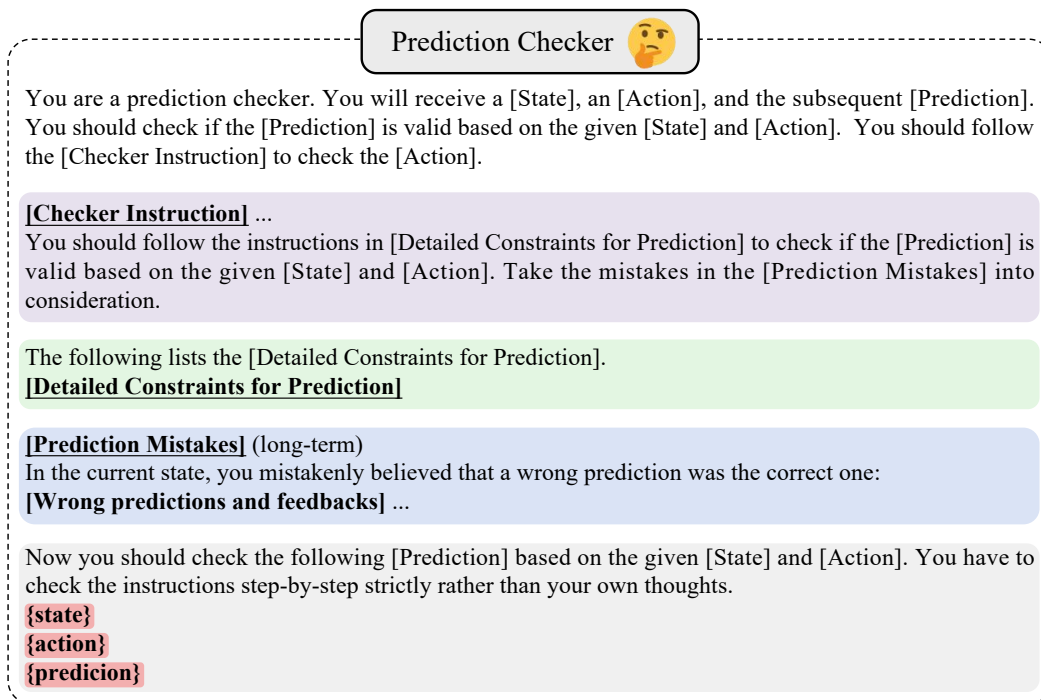


Figure 11: A simplified template for the prediction checker prompt.

I Detailed Experiment Results

I.1 Basic Policy Analysis for Multi-Agent Tasks

As mentioned in the experiment section, our MLAQ agent framework is compatible with both single-agent and multi-agent scenarios. In multi-agent scenarios, it provides two basic policy decision paradigms. The first one treats the entire multi-agent system as a whole for centralized decision-making, where it takes the global state as inputs and outputs joint actions for the system. The second one adopts RoCo’s dialog mechanism (Mandi et al., 2023), where the agents conduct multiple rounds of dialogue with each other and determine the joint actions when reaching consensus: $a_{i,t} \sim \pi_i(\cdot | o_{i,t}, \rho_{i,t})$, where $\rho_{i,t} = [d_{i,1}, \dots, d_{i-1,j}]$ is the chat history between agents. $d_{i,j}$ represents the message output by agent i at the dialog round j .

We evaluate the agents using these two basic policies in the Sort and Sandwich domains. As shown in Table 5, the results indicate that, with similar token consumption, the performance of centralized decision-making significantly decreases, with env replans increasing nearly tenfold in the Sandwich domain. The complexity of centralized decision-making significantly surpasses that of decentralized decision-making, thereby diminishing the effectiveness of basic policies in obtaining available or optimal actions.

Table 5: Performance comparison of using dialog and central paradigms in multi-agent domains.

Task	Methods	Success Rate	Optimal Rate	Token	Env Replans	Optimal Gap
Sort	dialog	0.98	0.75	175560	0.06	0.26
	central	0.88	0.59	199753	0.04	0.57
Sandwich	dialog	0.95	0.45	208024	0.07	0.61
	central	0.81	0.24	131244	0.62	3.24

I.2 Full Results in the Sort domain

Due to space limitations, the complete experimental results for the sort domain are provided in this section for reference.

Table 6: Comparison of multiple metrics between MLAQ and RoCo in Sort domain.

Metrics	Methods	1-step	2-step	3-step	4-step	5-step	6-step	Average
Success Rate	RoCo	1.00	0.64	0.47	0.10	0.03	0.00	0.35
	MLAQ	1.00	0.96	0.97	1.00	0.93	1.00	0.98
Env Replans (n-shot)	RoCo	0.30	6.30	5.60	9.74	7.67	6.92	6.41
	MLAQ	0.00	0.04	0.03	0.10	0.10	0.04	0.06
Optimal Rate	RoCo	0.80	0.36	0.27	0.00	0.00	0.00	0.21
	MLAQ	0.95	0.64	0.57	0.67	0.33	0.43	0.58
	MLAQ	1.00	0.86	0.77	0.73	0.50	0.75	0.75
Average Token	RoCo	10605	530817	332730	305175	345762	320045	322630
	MLAQ	7093	15104	16436	22197	18133	8220	15216
	MLAQ	8491	66367	156490	119916	409495	243151	175560
Optimal Gap	RoCo	0.35	3.37	3.43	3.33	2.97	2.00	2.72
	MLAQ	0.10	0.64	0.73	0.53	1.20	0.46	0.65
	MLAQ	0.00	0.32	0.40	0.43	0.80	0.25	0.39
Memory Re-Util. Ratio		0.27	0.61	0.71	0.72	0.73	0.83	0.66

I.3 Main Results in the Sandwich domain

Due to space limitations in the main text, we provide the experimental results in the Sandwich domain in this section for reference. As shown in Table 7, although MLAQ is superior to RoCo in success rate

and outperforms RoCo in long-horizon tasks (10-step), the token consumption of MLAQ is higher than that of RoCo. This is mainly due to two reasons. Firstly, the similarity between state transitions plays an essential role in token consumption, with tasks exhibiting higher similarity being better able to leverage stored transition data in memory to reduce token consumption. The Sandwich domain, with its relatively fixed trajectories compared to Sort, exhibits lower similarity across different tasks. This is evidenced by the average memory re-utilization ratio between the two domains, notably lower in Sort. This characteristic of the Sandwich domain not only contributes to RoCo achieving a higher optimal rate but also diminishes MLAQ’s token saving capability. Secondly, MLAQ itself involves world model prediction and self-examination, leading to an average token consumption per time step exceeding RoCo’s.

Table 7: Comparison of multiple metrics between MLAQ and RoCo in Sandwich domain.

Metrics	Methods	6-step	8-step	10-step	Average
Success Rate	RoCo	1.00	0.64	0.92	0.75
	MLAQ	1.00	0.92	1.00	0.95
Env Replans (n-shot)	RoCo	2.50	3.87	4.33	3.86
	MLAQ	0.00	0.11	0.00	0.07
Optimal Rate	RoCo	0.75	0.36	0.25	0.37
	MLAQ ⁻	0.00	0.08	0.83	0.29
	MLAQ	0.25	0.31	0.83	0.45
Average Token	RoCo	20740	35072	65027	42266
	MLAQ ⁻	74881	63555	77949	68746
	MLAQ	184609	220521	188752	208024
Optimal Gap	RoCo	0.50	1.13	1.83	1.27
	MLAQ ⁻	1.25	1.88	0.17	1.33
	MLAQ	0.75	1.35	0.17	0.96
Memory Re-Util. Ratio		0.31	0.63	0.57	0.58

I.4 MLAQ with Multiple Epochs

This experiment aims to demonstrate the potential of MLAQ to enhance its optimal decision-making ability through iterative optimization. In previous experiments, each domain’s experiment starts with an empty memory. In this section, we conduct multiple epochs of MLAQ experiments by performing MLAQ with a non-empty memory to further enhance its decision-making capabilities and utilize memory re-utilization to reduce token consumption. At the beginning of each epoch, the input memory is the memory of the last MLAQ epoch after finishing all given tasks of the domain. As shown in Table 8 and Figure 12, an additional training epoch results in an improvement of the optimal rate while reducing token consumption. Although MLAQ with multiple epochs no longer constitutes zero-shot decision-making, the small magnitude of env replans implies that minimal environmental information is injected into subsequent epoch decisions.

I.5 Self-Examination Analysis

This section quantitatively evaluates the LLM-based self-examination, showing that while hallucinations may occur during the examining process, performing multiple checks in transition imagination significantly reduces the erroneous transitions. In the self-examination process, the LLM-based checkers may erroneously identify correct outputs as incorrect, and vice versa. We sample 128

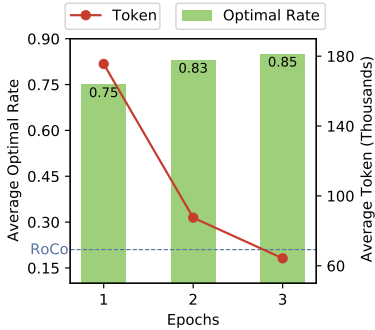


Figure 12: Tendency of optimal rates and tokens with epochs.

Table 8: The optimal rate for optimizing multiple epochs in Sort domain.

Methods	1-step	2-step	3-step	4-step	5-step	6-step
MLAQ-1st epoch	1.00	0.86	0.77	0.73	0.50	0.75
MLAQ-2nd epoch	1.00	0.93	0.90	0.77	0.57	0.88
MLAQ-3rd epoch	1.00	0.93	0.90	0.77	0.57	0.92

random states from the Sort domain, providing correct actions for 64 states and incorrect actions for the others, with each incorrect action breaking only one constraint. The testing for the prediction follows a similar approach, where each incorrect prediction only violates one constraint.

Table 9: Results on identification capability of the checkers.

Checker Type	TP \uparrow	FN \downarrow	FP \downarrow	TN \uparrow	Precision \uparrow
Action Checker	63	1	6	58	91.3%
Prediction Checker	64	0	10	54	86.5%

As shown in Table 9, we evaluate the performance of the checkers using five metrics. **True Positives (TP)** identify correct outputs (actions or predictions) as correct, **False Negatives (FN)** identify correct outputs as incorrect, **False Positives (FP)** identify incorrect outputs as correct, **True Negatives (TN)** identify incorrect outputs as incorrect, and **Precision = TP / (TP + FP)** is a proportion of correctly identified correct cases among all cases identified as correct. Results in Table 9 suggest that while both checkers accurately identify correct outputs, they may misidentify incorrect outputs as correct ones, leading to inaccuracies in imaginary transitions and subsequent impacts on agent performance.

To further investigate the impact of inaccuracies in self-examination, similar methods are used to test the accuracy of the basic policy and world model. Without forbidden actions, the accuracy of the basic policy is 84.4%, while the accuracy of the world model is 100%. Therefore, the combined calculation yields a probability of $1.36\% = (1 - 84.4\%) * [6 / (6 + 63)]$ for an wrong transition to be stored in memory. Compared to the probability of $15.6\% = (1 - 84.4\%)$ without self-examination, MLAQ can greatly improve the quality of imaginary transitions. Furthermore, some erroneous transitions stored in memory could be excluded from the best policy derived from RL-based optimization due to their low values, while the others could be corrected by env-examination. Experimental data in Table 2 also indicate that these errors minimally impact the decision-making capabilities of LLM agents.

J Hyper-Parameters

Table 10 shows the hyper-parameters presented in the MLAQ training process. Except for the environmental horizon T , all other hyper-parameters remain constant across all experiments, where T is set to 20 for the BlocksWorld experiment, 16 for the Sandwich experiment, and 8 for the Sort experiment.

In this paper, all experiments are conducted using the GPT API interface, without involving CPU or GPU usage. The total cost of the API resources used in this paper does not exceed 1500 US dollars, including preliminary tests, comparative experiments, and ablation experiments.


Table 10: Hyper-parameters presented in the MLAQ training process

Hyper Parameter	Value
LLM source	gpt-4-0125-preview
Learning rate α	1.0
Discount γ	0.995
UCB weight w_g	2
vUCB weight w_g	4
vUCB threshold ϵ_g	0
Maximum trial number for imagination K_c	2
Maximum trial number for prediction K_s	2
Maximum trial number for policy K_a	2
Q threshold \tilde{Q}	0.5
Q update loops	20
Environmental horizon T	20 / 8 / 16

K Prompt Example

K.1 Policy Prompt and Response of the State-based Decision-Making

In the LLM agent constructed by our method MLAG, both decision-making in single-agent scenarios and centralized decision-making in multi-agent scenarios rely on the global state. We use the prompt template in Figure 8 to query LLM and output agent actions or joint actions of the multi-agent system. In the following, we provide an example of a policy prompt in the Sort domain, which includes all the elements in the template shown.

Policy 

[Detailed Instruction for Policy]

You should output your thinking process step-by-step in the following order. You have to follow these steps one by one to plan the [Action]: [Action Planning], [Action Conclusion], [Action Constraints Check], [Forbidden Action Check], and [Action Output].

The most important thing: Follow the instructions step-by-step and ensure each step is completed precisely. Repeat the instructions and fill in the blanks '[]' without introducing any modifications or additional content.

- + **[Action Planning]**: Plan the [Action] of the multi-agent system step by step and list the thinking process.
- + **[Action Conclusion]**: Conclude the [Action] in the format of [Detailed Constraints for Action].
- + **[Action Constraints Check]**: Follow the steps in [Detailed Constraints for Action] to check the [Action] step by step.
- + **[Action Revise]**: If there is no incorrect checking result in the [Action Constraints Check], you can pass to the [Forbidden Action Check]. Otherwise, you should revise the action. Take the feedback from [Action Constraints Check] into consideration, and follow these steps to revise the action: [New Action Planning] and [NewAction Constraints Check].
- + **[Forbidden & Mistake Action Check]**: I will provide some forbidden joint actions, and the robots cannot choose the forbidden and mistake actions. Note that what is forbidden are the joint actions of the multi-agent system, rather than the single agent actions that appear in these joint actions.
 - list the chosen action.
 - list all actions in the [Forbidden Actions] and [Policy Mistakes].
 - check if the [Action] is in the [Forbidden Actions] or [Policy Mistakes] one by one.If the [Action] is not in the [Forbidden Actions] and [Policy Mistakes]:
 - Pass to the [Action Output] part.Else:
 - Follow these instructions to plan a new [Actions] step by step and list the thinking process:
 - Go back to a new [Action Planning] and [Action Constraints Check] process again. You should note that what is forbidden are the joint actions rather than the single agent actions that appear in these joint actions. Therefore, the individual actions of each agent in the [Forbidden Actions] can still be chosen.
 - [Forbidden Action Check]: check if the [Action] satisfies the [Detailed Constraints for Action].
 - Re-plan the [Action] until a new [Action] is obtained or there is no available action anymore. You can try 2-nd, 3-rd, 4-th ... Plan to get the final [Action].If there is no available action anymore, output [ERROR] to denote that there is no available action anymore.
- + **[Action Output]**: output the final action in the format of [Detailed Constraints for Action].

The following lists the [Task Description], [State Space Definition], and [Action Space Definition].

[Task Description]

Task: Cooperative Sorting in a Multi-Agent System

Agents: Alice, Bob and Chad

Alice - Can only PICK and PLACE cube items on panel1, panel2 and panel3.

Bob - Can only PICK and PLACE cube items on panel3, panel4 and panel5.

Chad - Can only PICK and PLACE cube items on panel5, panel6 and panel7.

Objective: Collaboratively place the cubes on the panels as follows: place blue_square on panel2, place pink_polygon on panel4, place yellow_trapezoid on panel6. The cube items will be placed randomly on all panels of the table at the beginning of an episode.

[State Space Definition]

Define the state of the multi-agent system, which is composed of one category: [Cube States].

1. [Cube States]: Describe the status of the following three cube items: blue_square, pink_polygon, yellow_trapezoid.
2. The cube items must be listed in this order: blue_square, pink_polygon, yellow_trapezoid.

[State Template]

[State]

[Cube States]

blue_square is on <location>

pink_polygon is on <location>

yellow_trapezoid is on <location>

[Detailed Constraints for State]

You should specify the [State] according to the following constraints. A state is reasonable if it does not break the constraints one-by-one and step-by-step.

1. There must be three lines of [Cube States].
2. Check if their locations are one of: on panel1 - 7.
3. It is not allowed to have two cubes on the same panel. Check if they satisfy constraints.
3. The panel name should not contain " " in the middle. For example, use "panel2" instead of "panel 2".

[Action Space Definition]

Detail the action space for each robot with the stipulated actions: PICK PLACE, WAIT.

- PICK <object> PLACE <target>: Execute if the gripper is empty. Pick up <object> and place it onto <target>, where <object> is a cube and <target> is a panel

- WAIT: Do nothing.

[Action Template]

EXECUTE

NAME Alice ACTION <action>

NAME Bob ACTION <action>

NAME Chad ACTION <action>

[Detailed Constraints for Action]

[Robot Reach Range] Before showing the constraints, you should note that each robot has its own reach range, and it cannot PICK or PLACE items outside its reach range.

+ Alice: panel1, panel2 and panel3

+ Bob: panel3, panel4, panel5

+ Chad: panel5, panel6, panel7

[Detailed Constraints] Then, you should follow the constraints one-by-one and step-by-step to check if the action is correct: [Basic Constraints] and [PICK & PLACE Constraints]. You have to follow these constraints strictly and do not have your own understanding of the constraints.

The most important thing: Follow the instructions step-by-step and ensure each step is completed precisely. Repeat the instructions and fill in the blanks '[' without introducing any modifications or additional content.

[Basic Constraints] Output "[Basic Check]" and do as follows:

- + There must be three lines of actions.
- + The agents cannot all perform WAIT.
- + Agents are not allowed to PICK <object> other than blue_square, pink_polygon, yellow_trapezoid.
- + Agents are not allowed to PICK the same <object> at the same time.
- + Agents are not allowed to PLACE objects on the same panel.

[PICK & PLACE Constraints] Now, output "[PICK & PLACE Check]" and do as follows:

- + For simplicity, we use [agent] to denote the agent's name: [Alice, Bob, Chad].
 - If action is PICK PLACE, follow the instructions step by step: [PICK Object Check], [PLACE Target Check].
 - a. [PICK Object Check] Agents are not allowed to PICK objects out of its reach range.
 - b. [PLACE Target Check] Agents are not allowed to PLACE objects out of its reach range.
 - c. [PLACE Target Check] Agents are not allowed to PLACE objects on the panel that already has an object except it is PICKed by one of the agents at the current timestep.
- + For example: List Alice's action ... List Bob's action ... List Chad's action ...''

[Policy Mistakes]

No policy mistakes.

[Temporary Mistakes]

You have made 2 mistakes. The following actions are the wrong actions you have made.

These feedbacks are from the previous imagination, and may not be correct. However, you still need to pay attention to them and avoid making the same or similar mistakes.

The wrong action 1 is as follows:

EXECUTE

NAME Alice ACTION WAIT

NAME Bob ACTION PICK yellow_trapezoid PLACE panel6

NAME Chad ACTION WAIT

The feedback is as follows:

[ERROR] The action is invalid because it violates the reach range constraint for Bob, who cannot PLACE the yellow_trapezoid on panel6 as it is outside of his range. Now, output your action below. You should follow the [Detailed Instruction for Policy] strictly and think step-by-step to plan the action.

The wrong action 2 is as follows:

```
EXECUTE
NAME Alice ACTION WAIT
NAME Bob ACTION PICK yellow_trapezoid PLACE panel5
NAME Chad ACTION WAIT
```

The feedback is as follows:

[ERROR] Bob cannot PICK the yellow_trapezoid from its current location (panel1) because it is outside his reach range (panel3, panel4, panel5). This violates the PICK Range Check constraint.

[Forbidden Actions]

The following lists the forbidden actions:

[Action 1]

```
EXECUTE
NAME Alice ACTION PICK yellow_trapezoid PLACE panel3
NAME Bob ACTION WAIT
NAME Chad ACTION WAIT
```

Now you should output your action according to the following current state and target state.

[State]

```
[Cube States]
blue_square is on panel2
pink_polygon is on panel4
yellow_trapezoid is on panel1
```

[Target State]

```
[Cube States]
blue_square is on panel2
pink_polygon is on panel4
yellow_trapezoid is on panel6
```

The response of the basic policy is shown below. Although it already outputs a forbidden action list in the response, it finally outputs an action in this list, which indicates that the response of LLM is actually imperfect. Therefore, our MLAQ agent try to optimize through RL methods.

The prompt and response in other tasks are also similar under the setting of state-based decision making.

[Action Planning]

- For Alice, since all her cubes are in the correct position, she should WAIT.
- For Bob, since all his cubes are in the correct position or being taken care of by Chad, he should WAIT.
- For Chad, since yellow_trapezoid needs to be moved from panel5 to panel6, and it's within his reach, he should PICK yellow_trapezoid PLACE panel6.

[Action Conclusion]

EXECUTE
NAME Alice ACTION WAIT
NAME Bob ACTION WAIT
NAME Chad ACTION PICK yellow_trapezoid PLACE panel6

[Action Constraints Check]**[Basic Check]**

- + There must be three lines of actions: [correct].
- + The agents cannot all perform WAIT: [correct].
- + Agents are not allowed to PICK <object> other than blue_square, pink_polygon, yellow_trapezoid: [correct].
- + Agents are not allowed to PICK the same <object> at the same time: [no agent has the same <object>, correct].
- + Agents are not allowed to PLACE objects on the same panel. [no agent has the same <target>, correct].

[PICK & PLACE Check]

- + For Chad:
 - Chad's action: [PICK yellow_trapezoid PLACE panel6].
 - a. [PICK Range Check] Agents are not allowed to PICK objects out of its reach range (Chad's: [panel5, panel6, panel7]): [no, action is incorrect].

[Action Revise]

Since the [PICK Range Check] for Chad's action was incorrect, the action needs to be revised. Chad cannot PICK the yellow_trapezoid from panel1 because it's out of his reach range. The correct action should involve Alice, who is in the range to PICK the yellow_trapezoid from panel1.

K.2 Policy Prompt and Response of the Observation-based Decision-Making

In the observation-based decision-making of LLM multi-agent systems, we fully utilize the RoCo's Policy Prompt to perform dialogues between agents, and only add an additional part to denote the forbidden actions. Here is an example in the Sort domain. We take the Alice's prompt as an example, the prompts of the other agents are also similar.

[New Action Planning]

Given the mistake identified during the [Action Constraints Check], we need Alice to move the yellow_trapezoid closer to Chad's range for him to eventually place it on panel6.

- Alice can PICK the yellow_trapezoid from panel1 and PLACE it on panel3, which is within her range and makes it accessible for Bob or Chad in subsequent actions.
- Bob and Chad will WAIT in this round to avoid any conflict.

[New Action Conclusion]

EXECUTE

NAME Alice ACTION PICK yellow_trapezoid PLACE panel3

NAME Bob ACTION WAIT

NAME Chad ACTION WAIT

[New Action Constraints Check]

[Basic Check]

- + There must be three lines of actions: [correct].
- + The agents cannot all perform WAIT: [correct].
- + Agents are not allowed to PICK <object> other than blue_square, pink_polygon, yellow_trapezoid: [correct].
- + Agents are not allowed to PICK the same <object> at the same time: [no agent has the same <object>, correct].
- + Agents are not allowed to PLACE objects on the same panel. [no agent has the same <target>, correct].

[PICK & PLACE Check]

- + For Alice's action: PICK yellow_trapezoid PLACE panel3.
 - a. [PICK Range Check] Agents are not allowed to PICK objects out of its reach range (Alice's: [panel1, panel2, panel3]): [correct].
 - b. [PLACE Range Check] Agents are not allowed to PLACE objects out of its reach range: [correct].
 - c. [PLACE Target Check] Agents are not allowed to PLACE objects on the panel that already has an object except it is PICKed by one of the agents at the current timestep: [The panel3 is empty, correct].

[Forbidden & Mistake Action Check]

- list the chosen action: EXECUTE NAME Alice ACTION PICK yellow_trapezoid PLACE panel3 NAME Bob ACTION WAIT NAME Chad ACTION WAIT
- repeat all actions and feedbacks in the [Forbidden Actions], [Policy Mistakes], and [Temporary Mistakes]:
 - a. [EXECUTE NAME Alice ACTION PICK yellow_trapezoid PLACE panel3 NAME Bob ACTION WAIT NAME Chad ACTION WAIT, forbidden action],
 - b. [EXECUTE NAME Alice ACTION WAIT NAME Bob ACTION WAIT NAME Chad ACTION WAIT, [ERROR] The action is invalid because all agents cannot perform WAIT simultaneously.],
 - c. [EXECUTE NAME Alice ACTION WAIT NAME Bob ACTION PICK yellow_trapezoid PLACE panel5 NAME Chad ACTION WAIT, [ERROR] Bob cannot PICK the yellow_trapezoid from its current location (panel1) because it is outside his reach range (panel3, panel4, panel5). This violates the PICK Range Check constraint.]]
- check if the [Action] is in the [Temporary Mistakes] one by one. If so, you should first judge whether the [Action] is correct or not. If the [Action] is correct, you can pass to the [Action Output] part. Otherwise, you should revise the [Action] step by step.
 - + The chosen action is not listed in the [Temporary Mistakes].
- check if the [Action] is in the [Forbidden Actions] one by one.
 - + The chosen action is not listed in the [Forbidden Actions].

[Action Output]

EXECUTE

NAME Alice ACTION PICK yellow_trapezoid PLACE panel3

NAME Bob ACTION WAIT

NAME Chad ACTION WAIT

Here we provide a dialogue result from the above scenario, where three agents reach the consensus on each other's decisions through two rounds of dialogue. Finally, Bob summarizes and outputs the final joint action.

The prompt and response in Sandwich domain are also similar under the setting of observation-based decision making.

Alice's Policy



[Action Space Definition]

- PICK <object> PLACE <target>: pick up <object> and place it onto <target>, where <object> is a cube and <target> is a panel
- WAIT: Do nothing.

[Action Template]

EXECUTE

NAME Alice ACTION <action>

NAME Bob ACTION <action>

NAME Chad ACTION <action>

[Action Output Instructions]

1. Commence the output with 'EXECUTE'
 2. Follow with one distinct action per robot on separate lines.
 3. Alice's action should be listed first, followed by Bob's action, and then Chad's action.
3. Agents' reach ranges are as follows:
- + Alice: zone1=[panel1, panel2, panel3]
 - + Bob: zone2=[panel3, panel4, panel5]
 - + Chad: zone3=[panel5, panel6, panel7]

You are robot Alice in front of panel2. You are collaborating with Bob, Chad to sort cubes into their target panels

Bob is tasked to place pink_polygon on panel4, and can reach panel3, panel4, panel5

Chad is tasked to place yellow_trapezoid on panel6, and can reach panel5, panel6, panel7

The task is NOT done until all three cubes are sorted correctly

There are 7 panels, ordered left to right: panel1,...,panel7. They form a straight assembly line, panel1 is closed to panel2 and farthest from panel7

Your goal is to place blue_square on panel2, but you can only reach panel1, panel2, panel3: this means you can only pick cubes from these panels, and can only place cubes on these panels

At current round:

blue_square is on panel5

pink_polygon is on panel4

yellow_trapezoid is on panel6

Never forget you are Alice!

1. You can only pick these reachable cubes:
 2. You can only place them on these reachable panels: panel1, panel2, panel3
- Think step-by-step about the task and others' response. Carefully check and correct them if they made a mistake
Improve your plans if given [Environment Feedback]

When you respond, tell others about your goal and all constraints. Respond very concisely but informatively, and do not repeat what others have said

Discuss with others to come up with the best plan, e.g. if your cube is out of your reach, ask others for help, and you can do the same for them

Propose exactly one action for yourself at the **current** round, select from [Action Options]

End your response by either: 1) output PROCEED, if the plans require further discussion, or 2) If everyone has made proposals and got approved, output EXECUTE and the final plan, must strictly follow [Action Output Instruction]!

You are Alice, before you make decisions, you should check if the [Action] is violating the [Forbidden Actions].

Your response is:

Since my target is to PUT ... on ..., and other robots ...

Dialogues



Alice - Round 1

Since my target is to place the blue_square on panel2, and other robots have their respective targets with Bob needing to place the pink_polygon on panel4 and Chad needing to place the yellow_trapezoid on panel6, we need to coordinate our actions. The blue_square is currently out of my reach on panel5, so I will need Bob's help to move it into my reach. Bob can pick up the blue_square from panel5 and place it on panel3, which is within both of our reach ranges. This action does not violate any forbidden actions as it involves moving a cube to an intersection panel to facilitate its transfer between zones. After Bob moves the blue_square to panel3, I can then pick it up and place it on its target panel, panel2, in a subsequent round.

Bob and Chad can proceed with their tasks as planned, with Bob moving the pink_polygon to panel4 as intended, and Chad placing the yellow_trapezoid on panel6. However, for this round, Bob's priority should be to assist me by moving the blue_square closer.

Let's discuss this plan and make sure it aligns with everyone's understanding and capabilities. If we all agree, we can proceed with the execution.

PROCEED

Bob - Round 1

Since my target is to place the pink_polygon on panel4, and other robots have their respective targets with Alice needing to place the blue_square on panel2 and Chad needing to place the yellow_trapezoid on panel6, we need to coordinate our actions carefully. The blue_square is currently on panel5, which is within my reach, and Alice has requested my help to move it into her reach. I can pick up the blue_square from panel5 and place it on panel3, which is within both Alice's and my reach ranges. This action does not violate any forbidden actions as it involves moving a cube to an intersection panel to facilitate its transfer between zones. After I move the blue_square to panel3, Alice can then pick it up and place it on its target panel, panel2, in a subsequent round.

My pink_polygon is already on panel4, which means it is correctly placed, and I do not need to move it this round. Chad can proceed with his task to place the yellow_trapezoid on panel6, which is already in the correct position as well.

Given Alice's plan and the current situation, my proposal for this round is to assist Alice by moving the blue_square closer to her. This will allow us to progress towards completing our overall task more efficiently.

Let's confirm if this plan works for everyone, including the movement of the blue_square to panel3 by me, and if there are no objections or further suggestions, we can proceed with the execution.

PROCEED

Chad - Round 1

Since my target is to place the yellow_trapezoid on panel6, and other robots have their respective targets with Alice needing to place the blue_square on panel2 and Bob needing to place the pink_polygon on panel4, we need to coordinate our actions carefully. The yellow_trapezoid is already correctly placed on panel6, which means my task for this round does not require any action on my part regarding the yellow_trapezoid.

Alice has proposed a plan for Bob to move the blue_square from panel5 to panel3, which is a feasible action and aligns with the goal of getting the blue_square closer to its target panel, panel2. This action does not violate any forbidden actions as it involves moving a cube to an intersection panel to facilitate its transfer between zones. Since my cube is already in the correct position, my focus this round should be on ensuring the plan aligns with the overall goal and does not interfere with my ability to maintain the yellow_trapezoid on panel6.

Bob has agreed to Alice's plan and has proposed to move the blue_square to panel3 to assist Alice. This action aligns with the collective goal of sorting all cubes into their target panels efficiently.

Given the current situation and the proposed actions by Alice and Bob, my action for this round will be to WAIT, as my cube is already in the correct position, and there are no immediate actions required from me to assist with the movement of the blue_square. My priority is to ensure the yellow_trapezoid remains on panel6 and to support Alice and Bob in their tasks as needed without violating any forbidden actions.

Let's confirm if this plan works for everyone, including my decision to WAIT this round, and if there are no objections or further suggestions, we can proceed with the execution.

PROCEED

Alice - Round 2

Since my target is to place the blue_square on panel2, and other robots have their respective targets with Bob needing to place the pink_polygon on panel4 and Chad needing to place the yellow_trapezoid on panel6, we need to coordinate our actions carefully. The blue_square is currently on panel5, which is out of my reach. I need Bob's assistance to move it into my reach. However, upon reviewing the constraints and forbidden actions, I realize that directly moving cubes to their target panels without violating any rules is our primary goal.

Bob can assist by moving the blue_square from panel5 to panel3, which is within both of our reach ranges. This action is crucial as it does not violate any forbidden actions; it involves moving a cube to an intersection panel (panel3) to facilitate its transfer between zones. After Bob moves the blue_square to panel3, I can then pick it up and place it on its target panel, panel2, in a subsequent round.

Bob's task of placing the pink_polygon on panel4 does not require immediate action since it's already in the correct position. Chad's task involves the yellow_trapezoid, which is also correctly placed on panel6. Therefore, Chad does not need to take any action regarding the yellow_trapezoid this round.

Given the current situation, my proposal for this round is for Bob to assist by moving the blue_square closer to me, while I wait for the blue_square to be within my reach. Chad can also wait since his cube is already in the correct position.

Let's confirm if this plan works for everyone. If there are no objections or further suggestions, we can proceed with the execution.

PROCEED

Bob - Round 2 (Conclusion)

Since my target is to place the pink_polygon on panel4, and other robots have their respective targets with Alice needing to place the blue_square on panel2 and Chad needing to place the yellow_trapezoid on panel6, we need to coordinate our actions carefully. The pink_polygon is already correctly placed on panel4, which means my task for this round does not require any action regarding the pink_polygon.

Alice has proposed that I assist by moving the blue_square from panel5 to panel3 to bring it within her reach. This action is within my capabilities and does not violate any forbidden actions, as it involves moving a cube to an intersection panel (panel3) to facilitate its transfer between zones. After moving the blue_square to panel3, Alice can then pick it up and place it on its target panel, panel2, in a subsequent round.

Chad's task involves the yellow_trapezoid, which is already correctly placed on panel6. Therefore, Chad does not need to take any action regarding the yellow_trapezoid this round and can wait.

Given the current situation and the proposed actions, my action for this round will be to move the blue_square from panel5 to panel3 to assist Alice. This will allow us to progress towards completing our overall task more efficiently. Since my pink_polygon is already in the correct position, and Chad's yellow_trapezoid is also correctly placed, there are no further actions required from me or Chad regarding our respective cubes.

Let's confirm if this plan works for everyone. If there are no objections or further suggestions, we can proceed with the execution.

EXECUTE

NAME Alice ACTION WAIT

NAME Bob ACTION PICK blue_square PLACE panel3

NAME Chad ACTION WAIT

K.3 Action Checker Prompt and Response Example

We present the prompt and response example of the action checker in a state action pair of the Sandwich domain.

Action Checker

You are an action checker. You should check if the [Action] is valid based on the given [State].

[Checker Instruction]

You should follow the check instructions in [Detailed Constraints for Action] to check the [Action] step-by-step. Take the mistakes in the [Action Mistakes] into consideration.

The following lists the [Task Description] and [Action Space Definition].

[Task Description]

Task: Cooperative Cooking in a Multi-Agent System

Agents: Chad and Dave

Chad - Can only PICK food items from the right side of the table.

Dave - Can only PICK food items from the left side of the table.

Objective: Collaboratively prepare a meal named "[bacon_sandwich]". The food items of target state must be assembled in the following sequence: bread_slice1, tomato, cheese, bacon, bread_slice2.

[Action Space Definition]

1) PICK <obj>: Pick one food <item>. Only PICK if gripper is empty. PICK only the correct next item according to the recipe

2) PUT <obj1> <obj2>: PUT <obj1> on top of <obj2>. <obj1> can be food, <obj2> can be food or cutting_board

3) WAIT, do nothing

Only one robot can PUT each round. You must PICK up an item before PUT

[Action Output Instruction]

Must first output 'EXECUTE\n', then give exactly one action per robot, put each on a new line

Dave can only pick up the food item on the left side of the table. Chad can only pick up the food item on the right side of the table

[Detailed Constraints for Action]

Check the following constraints and Fill in blanks in '[]'. Check these constraints one by one: 1, 2...

* Get [Extended Recipe Order]: [cutting_board, bread_slice1, ...] (cutting_board + [Recipe Order])

1. [Line Number Check] The line of the actions should be less than or equal to two.

2. [Agent Order Check] The first agent should be Chad and the second agent should be Dave.

3. [WAIT Number Check] The number of WAIT should be less than or equal to one.

4. [PUT Number Check] The number of PUT should be less than or equal to one.

5. [PICK Side Check] PICK the food item on the side of the robot.

6. [PICK Robot State Check] PICK the food item only if the gripper is empty.

7. [PICK Food State Check] PICK the food item only if the food item's state is not 'atop <obj>'.

8. [PUT Availability Check 1] If <obj2> is cutting_board, <obj1> must be bread_slice1.

9. [PUT Availability Check 2] If <obj2> is not cutting_board, state of <obj2> should equal to the food item before <obj2> in the [Extended Recipe Order].

10. [PUT Availability Check 3] If no other food item is 'atop <obj2>', this constraint is valid directly.

11. [PUT Order Check 1] <obj1> should equal to the first item in [Reachable Items].

12. [PUT Order Check 2] <obj2> should equal to the food item before <obj1> in [Extended Recipe Order].

13. [PICK Order Check] For PICK action, robot should PICK the first food item in its [Reachable Items].

[Action Mistakes]

No action mistakes.

Now you should check the following [Action] based on the given [State]

[State]

[Food States]

bread_slice1: atop cutting_board

bread_slice2: gripped by Dave

bacon: atop cheese

cheese: atop tomato

tomato: atop bread_slice1

```

cucumber: on right side
ham: on right side
beef_patty: on right side
[Robot States]
1. Chad's gripper is empty
2. Dave's gripper is holding bread_slice2

EXECUTE
NAME Chad ACTION WAIT
NAME Dave ACTION PUT bread_slice2 bacon

[Start]
[Extended Recipe Order]: ..., the first item in [Reachable Items]: ...
1.

```

```

[Extended Recipe Order]: [cutting_board, bread_slice1, tomato, cheese, bacon, bread_slice2].
1. [Line Number Check] The line of the actions should be less than or equal to two: [yes]
2. [Agent Order Check] The first agent should be Chad and the second agent should be Dave: [yes]
3. [WAIT Number Check] The number of WAIT should be less than or equal to one: [yes]
4. [PUT Number Check] The number of PUT should be less than or equal to one: [yes]
5. [PICK Side Check] PICK the food item on the side of the robot: [not applicable]
6. [PICK Robot State Check] PICK the food item only if the gripper is empty: [not applicable]
7. [PICK Food State Check] PICK the food item only if the food item's state is not 'atop <obj>': [not applicable]
8. [PUT Availability Check 1] If <obj2> is cutting_board, <obj1> must be bread_slice1: [not applicable]
9. [PUT Availability Check 2] If <obj2> is not cutting_board, state of <obj2> should equal to the food item before
<obj2> in the [Extended Recipe Order]: [yes]
10. [PUT Availability Check 3] Get <item>s in [State] whose state matches '<item>: atop bacon': pool=[]. If no
other food item is 'atop bacon' (meaning pool is empty), this constraint is valid directly: [yes]
11. [PUT Order Check 1] <obj1> should equal to the first item in [Reachable Items]: [yes]
12. [PUT Order Check 2] <obj2> should equal to the food item before <obj1> in [Extended Recipe Order]: [yes]
13. [PICK Order Check] For PICK action, robot should PICK the first food item in its [Reachable Items]: [not
applicable]

[Conclusion]
The [Action] is valid based on the given [State].

```

The action checker is based on state in both single-agent and multi-agent settings, so the checking process for all tasks is similar.

K.4 World Model Prompt and Response Example

We present the prompt and response example of the world model in a state action pair of the BlocksWorld task.

World Model

You are a state predictor. You will be given a [State] and an [Action] as input. should predict the [State] after the [Action] is applied to the given [State]. You should follow the [Predicting Instruction] to predict the [State].

[Predicting Instruction]

You will be provided with the [State] and the [Action] of the agent. You should think step by step to output the [Prediction] of the next [State] based on the given [State] and [Action]. The format of the [Prediction] should follow the [Detailed Constraints for State].

Please output your thinking process step-by-step by following these steps:

The most important thing: Follow the instructions step-by-step and ensure each step is completed precisely. Repeat the instructions and fill in the blanks '[]' without introducing any modifications or additional content.

- [Interaction Item Pool]: initialize the pool of the blocks that the agent is interacting with. It is a empty list at the beginning: pool={}.
+ The blocks that are not present in the pool should not change their state.
- [Action Forward Rule]: Follow these steps to predict the [Prediction]:
 - + If the action is "PICK UP" or "UNSTACK", the state of <object> in the [Prediction] should be "in hand". The state of your hand in the [Prediction] should be "Holding <object>". Add <object> to the pool: pool=[].
 - + If the action is "PUT DOWN", the state of <object> in the [Prediction] should be "on table". The state of your hand in the [Prediction] should be "Empty". Add <object> to the pool: pool=[].
 - + If the action is "STACK ON", the state of <object> in the [Prediction] should be "on <target>". The state of your hand in the [Prediction] should be "Empty". Add <object> to the pool: pool=[].
- [Prediction Conclusion]: Conclude the [Prediction] based on the [Action Forward Rule].
 - + The format of the [Prediction] should follow the [Detailed Constraints for State].
 - + The blocks that are not present in the pool should not change their state.

The following lists the [State Space Definition].

[State Space Definition]

Define the state of the scenarios: blocks and hand.

- [Hand State]: The state of the hand. It can be "Empty" or "Holding <object>", where <object> is a block.
- [Block States]: The state of each block. It can only be one of ["on <object>", "on table", "in hand"]. The blocks should be listed in this order: blue block, orange block, red block, yellow block.

[State Template]

[State]

[Hand State]

<hand state>

[Block States]

blue block: <block state>

orange block: <block state>

red block: <block state>

yellow block: <block state>

[Detailed Constraints for State]

The following paragraphs are the detailed constraints for State. There are three main <check items>: [Title Check], [Hand State Check], and [Block States Check].

- [Title Check] A header titled [State].
- [Hand State Check] A header titled [Hand State]. Then, list the Hand State.
 - + Hand state must be one of ["Empty", "Holding <object>"], where <object> is a block (e.g. "Holding blue block").
- [Block States Check] A header titled [Block States]. Then, list the states of each block.
 - + Each line represents the state of one block item.
 - + The block items must be listed in this order: blue block, orange block, red block, yellow block.
 - + The <block state> specifies the state of the block. It can only be "on <object>", "on table" or "in hand" (e.g. "blue block: on table", "orange block: on red block", "red block: in hand").

[Prediction Mistakes]

No prediction mistakes.

Now, you should predict the next [State] after the following [Action] is applied to the given [State]. You have to follow the instructions strictly rather than your own thoughts.

The response of the world model is relatively concise, as it only needs to infer the next state according to the given instructions.

[State]
[Hand State]
Holding blue block
[Block States]
blue block: in hand
orange block: on yellow block
red block: on orange block
yellow block: on table

EXECUTE
STACK blue block ON red block

[Interaction Item Pool]: initialize the pool of the blocks that the agent is interacting with. It is an empty list at the beginning: pool={}.
[Action Forward Rule]: List the action: [STACK blue block ON red block], the state in the [Prediction] is changed by the [Action].
- If the action is "STACK ON", list the <object> of action: [blue block], and <target> of action: [red block]. The state of <object> in the [Prediction] should be changed to "on red block". The state of your hand in the [Prediction] should be changed to "Empty". Add <object> to the pool: pool=[blue block].

[Prediction Conclusion]:
[State]
[Hand State]
Empty
[Block States]
blue block: on red block
orange block: on yellow block
red block: on orange block
yellow block: on table

K.5 Prediction Checker Prompt and Response Example

We present the prompt and response example of the prediction checker in a state action pair of the Sort domain.

Prediction Checker 🤔

You are a prediction checker. You will receive a [State], an [Action], and the subsequent [Prediction]. You should check if the [Prediction] is valid based on the given [State] and [Action]. You should follow the [Checker Instruction] to check the [Action].

[Checker Instruction]

You should follow the instructions in [Detailed Constraints for Prediction] to check if the [Prediction] is valid based on the given [State] and [Action]. Take the mistakes in the [Prediction Mistakes] into consideration.

The following lists the [Detailed Constraints for Prediction].

[Detailed Constraints for Prediction]

Check the following constraints and Fill in blanks in '[]'. Check these constraints one by one: 1, 2... Only get your conclusion according to the results of these checks!

* Get [Interaction Item Pool]: list all PICKed objects in the [Action]: <pool>=[].

1. [Header Check] A header titled [State] -> A header titled [Cube States] -> List the Cube States.
2. [Cube Order Check] Cubes must be listed in the following order: [blue_square, pink_polygon, yellow_trapezoid].
3. [Cube Format Check] Each line should be formatted as "<cube_name> is on <panel_name>". Three lines in total.
4. [Cube State Check] Don't have two cubes on the same panel.
5. [PICK & PLACE Rule Check] For PICK & PLACE, state of <object> should be '<object> is on <target>'.

* When checking, you should repeat the prompt and fill in blanks, replace <obj> and <agent> with their true values, and give the line conclusion (yes/no/not applicable). For example:

4. [Cube State Check] Repeat cubes' locations: [panel1, panel2, panel3]. Don't have two cubes on the same panel: [yes]. Line conclusion: [yes]

* Constraint 5 may have multiple agents of checking. If so, you should output results separately. For example:

5.1 [PICK & PLACE Rule Check] For Alice's action: [PICK blue_square PLACE panel2], repeat <object>=[blue_square], <target>=[panel2], state of <object> in the prediction: [blue_square is on panel2], it should be '<object> is on <target>': [yes]. Line conclusion: [yes]

5.2 [PICK & PLACE Rule Check] For Bob's action: ... 5.3 ...

[Prediction Mistakes]

No prediction mistakes.

Now you should check the following [Prediction] based on the given [State] and [Action]. You have to check the instructions step-by-step strictly rather than your own thoughts.

The original state is:

[State]

[Cube States]

blue_square is on panel6

pink_polygon is on panel4

yellow_trapezoid is on panel2

The action is:

EXECUTE

NAME Alice ACTION PICK yellow_trapezoid PLACE panel1

NAME Bob ACTION WAIT

NAME Chad ACTION PICK blue_square PLACE panel5

The prediction is:

[State]

[Cube States]

blue_square is on panel5

pink_polygon is on panel4

yellow_trapezoid is on panel1

[Start]

Get [Interaction Item Pool]: list all ...

1. [Header Check]

2. [Cube Order Check] ...

[Conclusion] ...

Due to passing all the checks, this prediction is considered correct.

[Start]

Get [Interaction Item Pool]: list all PICKed objects in the [Action]: <pool>=[yellow_trapezoid, blue_square].

1. [Header Check] A header titled [State] -> A header titled [Cube States] -> List the Cube States: [yes].
2. [Cube Order Check] Cubes must be listed in the following order: [blue_square, pink_polygon, yellow_trapezoid]: [yes].
3. [Cube Format Check] Each line should be formatted as "<cube_name> is on <panel_name>". Three cube lines in total: [yes].
4. [Cube State Check] Repeat cubes' locations: [panel5, panel4, panel1]. Don't have two cubes on the same panel: [yes]. Line conclusion: [yes].
5. [PICK & PLACE Rule Check]
 - 5.1 For Alice's action: [PICK yellow_trapezoid PLACE panel1], repeat <object>=[yellow_trapezoid], <target>=[panel1], state of <object> in the prediction: [yellow_trapezoid is on panel1], it should be '<object> is on <target>': [yes]. Line conclusion: [yes].
 - 5.2 For Bob's action: [WAIT], no object interaction to check. Line conclusion: [not applicable].
 - 5.3 For Chad's action: [PICK blue_square PLACE panel5], repeat <object>=[blue_square], <target>=[panel5], state of <object> in the prediction: [blue_square is on panel5], it should be '<object> is on <target>': [yes]. Line conclusion: [yes].

[Conclusion] All checks are valid based on the provided [Action] and the resulting [Prediction]. The prediction is accurate.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: The abstract and introduction could accurately reflect the paper's contributions and scope.

Guidelines:

- The answer NA means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: Please refer to the Appendix E for more details.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory Assumptions and Proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: There are no theoretical results in this work.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental Result Reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: In this paper, we provide a detailed description of the framework's components and present comprehensive pseudo-codes to illustrate the overall process. Additionally, we have anonymously open-sourced the algorithm's code, allowing users to freely adjust hyper-parameters and utilize two types of the basic policy (central and dialogue). Furthermore, we have included a link in the aforementioned repo to present all experimental results, including all prompts and LLM responses, to thoroughly demonstrate the reproducibility of our method.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We have open-sourced the algorithm’s code anonymously on GitHub. We also provide all experimental results in this repo, including prompts and the corresponding LLM responses.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so “No” is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://nips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental Setting/Details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: The experimental details are provided in the main text and Appendix F. We also provide the hyper-parameters in Table 10.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment Statistical Significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: We do not provide error bars primarily due to the high cost of a single experiment. However, we evaluate the algorithm’s performance across multiple domains and provide the code and experimental data, aiming to sufficiently demonstrate the reliability of the performance to the reviewers.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments Compute Resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The only compute resource used in this work is the GPT API interface. Please refer to Appendix J for more details.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code Of Ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines?>

Answer: [Yes]

Justification: We confirm that this research complies with the NeurIPS Code of Ethics.

Guidelines:

- The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader Impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: Although this paper proposes an LLM-based agent framework, we utilize it to only address decision-making tasks without updating the LLM parameters. Therefore, we do not consider it to have any societal impacts.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification: This research does not poses such risks.

Guidelines:

- The answer NA means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [NA]

Justification: This paper does not use existing assets.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.

- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification: This paper does not release new assets.

Guidelines:

- The answer NA means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: This paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification: This paper does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.