DDCG: Decoupled Dual-Critic Guidance for Embodied Agents

Shaojin Ma, Min Zhang, Hongyao Tang, Jianye Hao*, Yan Zheng
College of Intelligence and Computing, Tianjin University

Abstract

Large Language Models (LLMs) have endowed embodied agents with unprecedented high-level planning capabilities. However, grounding abstract language plans into the physical world remains a significant challenge. Although feedback-based closed-loop systems are the dominant paradigm, we identify a critical bottleneck: **Signal Confounding**. Current feedback mechanisms fail to distinguish between physically infeasible errors, which arise from violating physical rules, and strategically sub-optimal choices. This ambiguity severely hinders effective plan correction. To address this, we propose **Decoupled Dual-Critic Guidance (DDCG)**, a framework that guides planning by providing two independent and explicit feedback signals. DDCG utilizes: a **Feasibility Critic** (C_F) to judge whether an action is physically compliant, and a **Quality Critic** (C_Q) to evaluate the strategic value of an action, conditioned on its feasibility. This decoupled guidance enables the LLM planner to perform precise error attribution, leading to better decision-making. Theoretically, DDCG can be viewed as a form of guided planning under dual-critic constraints: the Feasibility Critic defines a hard *safety* boundary, while the Quality Critic provides a *reward* gradient for guidance within it. This allows for more effective planning without requiring expensive parameter updates. Extensive experiments on the embodied benchmark VirtualHome demonstrate that DDCG significantly improves both task success rate and plan executability, establishing a more robust new paradigm for the LLM grounding problem.

1 Introduction

In recent years, with the rapid evolution and widespread application of Large Language Models (LLMs), they have shown unprecedented potential in the field of embodied artificial intelligence. In particular, significant breakthroughs in core capabilities such as reasoning and planning have brought widespread attention to the application prospects of LLMs in the field of embodied task planning (i.e., decomposing complex tasks into effective, executable steps for an agent). However, embodied tasks in the real world are often constrained by complex physical rules and logical relationships, which pose a great challenge for task planners to generate executable solutions. Recent studies show that even for state-of-the-art models like GPT-4[12], the generated plans still commonly have problems such as being physically unachievable, logically incoherent, or factually incorrect [8, 18]. This highlights the urgent need to improve the reliability of task planning for agents. [5, 3].

Currently, LLM-based task planners are mainly divided into two paradigms: open-loop planning and closed-loop planning[2]. The open-loop planning paradigm usually refers to the model generating a complete planning sequence at once without environmental interaction; while this paradigm is simple to implement, it struggles to handle dynamic environmental changes and complex constraints. Thus, the current mainstream paradigm has shifted to closed-loop planning, whose core mechanism

*Corresponding author: Jianye Hao(jianye.hao@tju.edu.cn)

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: LAW.

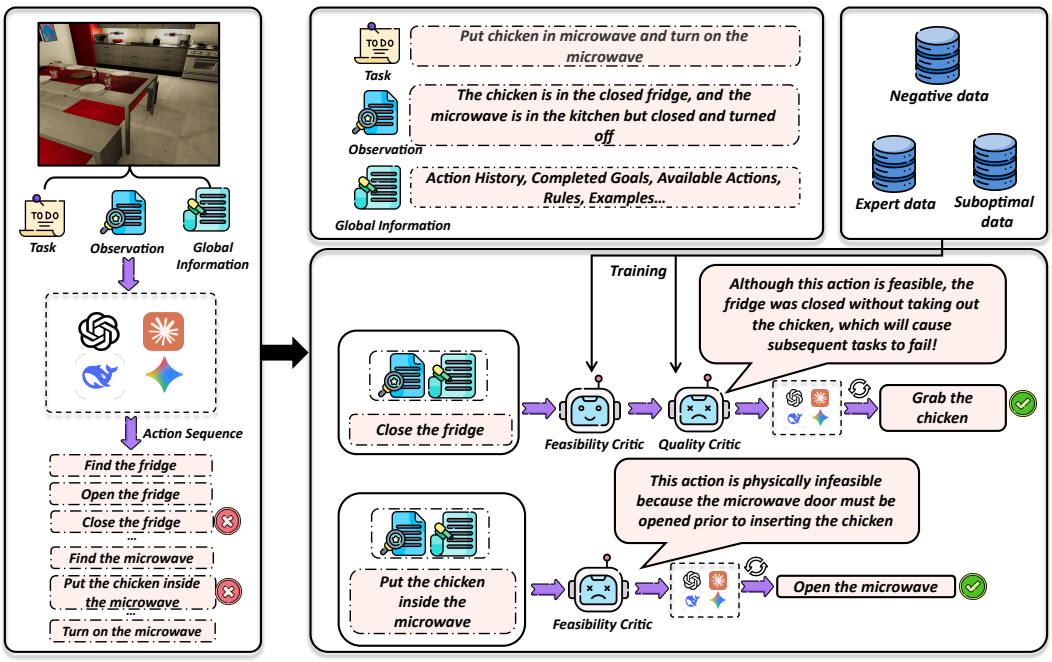

Figure 1: An overview of our **D**ecoupled **D**ual-**C**ritic **G**uidance (DDCG) pipeline. Given a high-level task, a base LLM planner generates a preliminary action plan that often suffers from "Signal Confounding," where infeasible and suboptimal actions are intertwined. Our DDCG framework intercepts this plan and decouples the verification into two sequential stages. First, the Feasibility Critic (C_F) acts as a learned safety constraint, pruning any action that is physically or logically impossible. Second, the Quality Critic (C_Q) evaluates the remaining feasible actions based on their strategic merit, filtering out those that are inefficient for the task goal. This two-stage process ensures that only actions that are both feasible and of high quality are executed, leading to a robust and successful final plan.

requires the agent to maintain dynamic interaction with the environment and perform self-reflection and continuous optimization based on the interaction results. According to the difference in feedback mechanisms, the closed-loop planning paradigm can be further divided into two categories: the self-feedback mechanism, with representative methods such as ReAct [22] and Inner Monologue [9], which achieves self-reflection and error correction by integrating environmental interaction results with experience memory, thereby improving the accuracy of subsequent planning [16, 11, 14, 15]; and the external feedback mechanism, where an external knowledge source provides more fine-grained, progressive guidance [23]. Existing guidance methods mainly include two forms: one is providing a single numerical score feedback for decomposed actions through a well-trained discriminator or reward model [6, 20, 4]; the other is using another powerful LLM to generate natural language feedback for decomposed actions [21].

Overall, the aforementioned methods significantly improve the accuracy of task decomposition through self-feedback or external feedback mechanisms, making substantial progress compared to traditional methods. However, existing methods commonly face a key bottleneck—the ambiguity of the feedback signal. Specifically, these methods have not yet precisely fed back the reasons for errors in task decomposition steps, such as: **feasibility errors** (violating hard rules) and **quality errors** (a poor strategic choice). This ambiguity directly leads to the LLM's inefficient decomposition.

Therefore, we propose a Decoupled Dual-Critic Guidance (DDCG) framework to address the ambiguity of the feedback signal by explicitly decomposing the guidance process. We have specifically designed two specialized, lightweight critics. The first critic is the **Feasibility Critic** (C_F), which learns a model of the environment's hard constraints to prune the action space to a physically and logically feasible range by verifying physical rule compatibility and logical consistency. The second critic is the **Quality Critic** (C_Q), which learns a heuristic value function to evaluate the value of

the current action from dimensions such as task goal achievement and resource efficiency. The proposed decoupled architecture is grounded in the principles of constrained optimization. The Feasibility Critic's role is conceptually analogous to that of a learned cost function in a Constrained Markov Decision Process (CMDP)[1], defining a "safe" action subspace . Our contributions can be summarised as follows:

- We are the first to identify and formally define "Signal Confounding" as a key obstacle that spans multiple existing paradigms in LLM-based embodied planning.
- We propose DDCG, a novel framework that resolves this issue by explicitly decoupling the evaluation of action feasibility and strategic quality, providing a mechanism for precise error attribution and targeted plan optimization.
- We demonstrate through experiments on the VirtualHome benchmark [13] that DDCG is projected to achieve state-of-the-art (SOTA) performance in both plan executability and task success rate.

2 Preliminary

Problem formulation as a CMDP. We formalize the embodied planning task as a Constrained Markov Decision Process (CMDP)[1], defined by the tuple $(\mathcal{S}, \mathcal{A}, P, R, C, \gamma)$. Here, $\mathcal{S}$ is the state space (textual descriptions of the environment), $\mathcal{A}$ is the action space (textual descriptions of possible actions), $P(s'|s,a)$ is the transition probability, $R(s,a)$ is the reward function, $C(s,a)$ is a cost function, and γ is the discount factor. The goal of the agent is to learn a policy π that maximizes the expected cumulative reward while ensuring the expected cumulative cost is below a certain threshold d:

$$\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)\right] \text{ s.t. } \mathbb{E}_{\pi}\left[\sum_{t=0}^{T} \gamma^t C(s_t, a_t)\right] \leq d, \tag{1}$$

We assume that any infeasible action (violating physical or logical rules) will lead to catastrophic failure. Therefore, the cost function is infinite for any infeasible action and zero otherwise. To enforce strict safety constraints and ensure that no infeasible actions are taken, we set a cost threshold $d = 0$. Therefore, our optimization goal is to find a policy that maintains an expected cumulative cost of zero, which is only possible if the policy never selects an action with a non-zero probability of being infeasible.

3 Method

DDCG tackles this CMDP by decomposing the policy guidance into two distinct modules that handle the constraint and the reward optimization separately.

3.1 Theoretical Analysis

We can formalize the DDCG framework as a critic-regularized optimization problem, aiming to improve a base LLM planner policy, π^{llm}. Our two critics provide distinct signals: the Quality Critic (C_Q) offers a reward signal, while the Feasibility Critic (C_F) imposes a hard constraint on the action space.

Let's define an implicit reward function based on the score from our C_Q: $r_Q(s,a) = C_Q(s,a)$. The C_F defines a safe action set $\mathcal{A}_{\text{safe}}(s) = \{a \in \mathcal{A} \mid C_F(s,a) = 1\}$.

The objective of DDCG is to derive an improved policy, π^{ddcg}, that maximizes the expected reward from the Quality Critic, while remaining close to the original LLM's policy distribution and strictly adhering to the safety constraints. As shown in Equation 2, this can be formulated as a constrained optimization problem:

$$\pi^{ddcg} = \arg\max_{\pi} \mathbb{E}_{a\sim\pi(\cdot|s)}[r_Q(s,a)] - \beta D_{KL}[\pi(\cdot|s)||\pi^{llm}(\cdot|s)], \tag{2}$$

subject to the hard constraint that $\pi(a|s) = 0$ for all $a \notin \mathcal{A}_{\text{safe}}(s)$. Here, β is a regularization coefficient that controls the trade-off between maximizing the quality score from the critic and staying

close to the LLM's original policy distribution. A smaller β encourages more aggressive optimization towards the critic's rewards, while a larger β ensures the resulting policy does not deviate drastically from the planner's initial knowledge. The value of β can be tuned as a hyperparameter on a validation set.

Following the derivation for critic-regularized RL [6], the optimal policy π^* that solves this problem has a specific analytical form.

Lemma 3.1. The optimal policy that solves the constrained optimization problem in Equation 2 is given by:

$$\pi^*(a|s) = \begin{cases} \frac{1}{Z(s)}\pi^{llm}(a|s)\exp(r_Q(s,a)/\beta) & \text{if } a \in \mathcal{A}_{\text{safe}}(s) \\ 0 & \text{if } a \notin \mathcal{A}_{\text{safe}}(s) \end{cases}, \tag{3}$$

where $Z(s) = \sum_{a' \in \mathcal{A}_{\text{safe}}(s)} \pi^{llm}(a'|s)\exp(r_Q(a'|s)/\beta)$ is a normalization factor.

This form provides a theoretical grounding for our DDCG workflow. The LLM planner first proposes actions according to its original policy π^{llm}. The Feasibility Critic (C_F) then applies the hard constraint, effectively restricting the action space to $\mathcal{A}_{\text{safe}}(s)$. Finally, the Quality Critic (C_Q) provides the reward signal r_Q, and our framework selects the action with the highest value, which is equivalent to sampling from the re-weighted distribution defined by π^*. This leads to a direct corollary on policy improvement.

Corollary 3.2. The updated policy $\pi^*(a|s)$ is a guaranteed improvement over the base policy $\pi^{llm}(a|s)$ with respect to the Q-function defined by our Quality Critic, i.e., $Q^{\pi^*}(s,a) \geq Q^{\pi^{llm}}(s,a)$.

The proof is a direct adaptation of the policy improvement theorem within a regularized MDP context. This analysis formally demonstrates that our DDCG framework does not merely filter actions, but guides the LLM towards a provably better policy distribution that is both safer (via C_F) and of higher quality (via C_Q).

3.2 Decoupled Dual-Critic

Feasibility Critic (C_F): A learned safety constraint The Feasibility Critic, C_F, is a function $C_F : \mathcal{S} \times \mathcal{A} \to \{0, 1\}$ that learns to approximate the environment's hard constraints. An action is deemed infeasible if it violates the environment's physical rules (e.g., attempting to place an object without holding it) or logical preconditions (e.g., trying to open an already open door). The critic takes the current state s and a candidate action a to predict its feasibility. This critic effectively learns to identify the safe action set $\mathcal{A}_{\text{safe}}(s) = \{a \in \mathcal{A} \mid C_F(s,a) = 1\}$. By filtering out any action $a \notin \mathcal{A}_{\text{safe}}(s)$, the C_F acts as a hard constraint satisfaction module. We implement C_F using a RoBERTa-based binary classifier.

Quality Critic (C_Q): An implicit reward function Once an action is certified as feasible by the C_F, the Quality Critic, C_Q, evaluates its strategic merit. The C_Q is a single regression model that takes a state-action pair and outputs a score, serving as a dense, implicit reward signal: $r_Q(s,a) = C_Q(s,a)$. This model is trained on both expert data (labeled with a score of 10) and the synthetically generated suboptimal data (labeled with scores from 3-9). The training objective is to minimize the Mean Squared Error (MSE) between the model's prediction and the ground-truth scores. This process enables the C_Q to learn a continuous utility gradient within the safe action space, guiding the planner towards action sequences that efficiently complete the task.

3.3 Data generation for a decoupled architecture

Training two specialized critics—a binary classifier for feasibility and a regressor for quality—requires a structured and diverse dataset that a simple collection of expert trajectories cannot provide. Our data generation pipeline is therefore uniquely designed to produce distinct signals tailored to each critic's learning objective. We synthesize three types of data:

1. **Expert Data (Score = 10):** This dataset consists of perfect, optimal action sequences from ground-truth solutions. It provides the gold standard for both critics, representing actions that are both maximally feasible and of the highest quality.

Algorithm 1 DDCG Online Planning

Require: LLM Planner, trained C_F, trained C_Q, threshold τ.

Given current state s and goal g.
loop
  Propose candidate action $a \leftarrow$ LLM-Planner(s, g).
  // Feasibility Check
  if $C_F(s, a) == 0$ **then**
    A feedback signal indicating infeasibility is delivered to the LLM.
    continue
  end if
  // Quality Check
  $score_{CQ} \leftarrow C_Q(s, a)$.
  if $score_{CQ} \geq \tau$ **then**
    Execute action a in the environment.
    break
  else
    A feedback signal indicating low quality is delivered to the LLM.
  end if
end loop

2. **Hard Negative Samples (Score < 3):** This is a key component designed specifically for the Feasibility Critic (C_F). Unlike simple random negatives, these samples are generated by applying rule-based perturbations to expert actions, creating actions that are explicitly infeasible. The perturbations include violating preconditions (e.g., attempting to 'slice' an object before it is held) or breaking action syntax. This dataset is crucial for teaching the C_F to recognize and penalize violations of the environment's hard physical and logical rules, a distinct requirement not present in single-critic frameworks like DGAP [6].
3. **Suboptimal Data (Score 3-9):** To effectively train the Quality Critic (C_Q) as a fine-grained regressor, we generate a large corpus of actions that are feasible but vary in strategic value. We use a fine-tuned FLAN-T5 model with beam search to produce a wide range of candidate actions. While the generation mechanism is related to prior work [8], our objective is twofold and tailored to our decoupled architecture: (1) to provide a rich, continuous spectrum of scores (from 3 to 9, assigned via semantic similarity to the expert action) that is essential for the C_Q's regression task, and (2) to simultaneously serve as a large and diverse set of positive examples for training the C_F, teaching it to distinguish valid actions from the hard negatives.

This tripartite data strategy directly supports our decoupled architecture. It enables the C_F to learn a strict decision boundary for what is possible, while allowing the C_Q to learn a nuanced utility function over what is strategically preferable, all without requiring an exhaustive amount of expert data.

3.4 Online planning

The online planning workflow, detailed in Algorithm 1, directly implements our "filter-then-evaluate" philosophy. At each step, the LLM planner proposes a candidate action. This action is first screened by the C_F. If deemed infeasible, a corresponding feedback signal is generated, prompting the planner to reconsider. If the action passes this initial check, it is then evaluated by the C_Q. Only if the action's quality score exceeds a predefined threshold τ is it finally executed in the environment.

4 Experiments

4.1 Experimental Setting

In this section, we detail the experimental setup designed to validate our proposed DDCG framework. We outline the benchmark environment, the metrics used for evaluation, the baselines selected for comparison, and the implementation and training specifics of our method.

Benchmark. We evaluate our method on VirtualHome [13], a challenging 3D simulation for embodied agents that requires long-horizon planning amidst complex object interactions. To ensure a standardized comparison, our evaluation follows the established dataset splits from prior work [10], which include three settings designed to test generalization: *In-Distribution*, *Novel Scenes*, and *Novel Tasks*.

Evaluation Metrics. We use two standard metrics for this benchmark [6]. **Success Rate (SR)** is a task-level metric that evaluates the final outcome of the entire plan. In contrast, **Executability (Exec)** is a step-level metric that measures the fraction of actions in a generated plan that are valid and can be executed by the environment.

Implementation and Training Details. For our method, DDCG, we use **GPT-4[12]** as the backbone LLM planner. The critic models, C_F and C_Q, are implemented as lightweight, fine-tuned RoBERTa-base models. The Feasibility Critic (C_F) is trained as a binary classifier on our entire synthesized dataset using a Binary Cross-Entropy loss. The Quality Critic (C_Q) is trained as a regressor exclusively on the feasible data using a Mean Squared Error loss. Key hyperparameters for critic training are detailed in Table 1.

Table 1: Hyperparameters for Critic Model (C_F and C_Q) Training.

Hyperparameter	**Feasibility Critic (C_F)**	**Quality Critic (C_Q)**
Model Architecture	RoBERTa-base	RoBERTa-base
Task Type	Classification	Regression
Max Sequence Length	256	256
Learning Rate	1e-5	1e-5
Optimizer	AdamW	AdamW
Weight Decay	0.01	0.01
Warmup Ratio	0.1	0.1
Batch Size	32	32
Training Epochs	5	5
Seed	42	42

4.2 Experimental Results

As shown in Table 2, our method not only achieves a leading performance in success rate(SR) but also shows a clear advantage in executability(Exec). Compared to all baselines, our method's SR and Exec values are highly consistent, which reveals the effectiveness of our decoupled approach in resolving the "Signal Confounding" problem.

On **In-Distribution** tasks, our method achieves a 95.0% success rate, significantly outperforming the strongest baseline, DGAP-GPT4 (88.0%). This advantage is magnified in the generalization settings. On **Novel Scenes** and **Novel Tasks**, we maintain high success rates of 73.6% and 81.8%, respectively. This robust performance is best understood by contrasting it with methods that excel at plan validity but fail strategically. For instance, Tree Planner [7] achieves high executability (90.3%) on novel tasks but suffers from poor strategic coherence, leading to a much lower success rate (52.3%). This gap between high executability and low success is a direct manifestation of the "Signal Confounding" problem, demonstrating that ensuring feasibility alone is insufficient for success in unfamiliar scenarios.

We attribute the advantage of our method in generalization to the successful resolution of the "Signal Confounding" problem. By decoupling feasibility (C_F) from quality (C_Q), our framework not only eliminates the gap between executability and success rate but also provides the necessary safety and strategic guidance to achieve superior performance in novel tasks and scenes.

4.3 Ablation Study

To validate the necessity of both the Feasibility Critic (C_F) and the Quality Critic (C_Q), we conducted ablation studies on the VirtualHome benchmark, comparing our full DDCG framework against two variants: **DDCG w/o** C_F (only quality guidance) and **DDCG w/o** C_Q (only feasibility filtering). As shown in 3, removing either critic leads to a significant performance degradation. Removing

Table 2: Main results on the VirtualHome benchmark. DDCG significantly outperforms all baselines, demonstrating the effectiveness of decoupled criticism.

	In-Distribution		Novel Scenes		Novel Tasks	
Method	**Exec. (%)**	**SR (%)**	**Exec. (%)**	**SR (%)**	**Exec. (%)**	**SR (%)**
MLDT[19]	38.0	34.0	32.0	31.0	34.0	33.0
LID-ADG[10]	-	46.7	-	32.2	-	25.5
ProgPrompt[17]	87.3	82.3	38.7	32.3	49.7	49.0
Tree Planner[7]	-	-	**89.3**	41.7	**90.3**	52.3
Inner Monologue[9]	79.7	79.3	54.3	53.3	47.3	46.0
DGAP-GPT4[6]	93.3	88.0	71.7	62.7	73.7	72.2
DDCG-GPT4(Ours)	**95.0**	**94.1**	73.6	**75.6**	83.3	**81.8**

C_F causes a substantial drop in Executability (e.g., from 95.0% to 89.4% In-Distribution) and an even sharper decline in Success Rate (from 94.1% to 82.3%), confirming that C_F is essential for ensuring plan validity. Conversely, removing C_Q results in a major degradation in SR, especially in the challenging *Novel Scenes* setting (from 73.6% to 63.2%), demonstrating that feasibility alone is insufficient for effective planning. Without the strategic guidance of C_Q, the agent often executes valid but inefficient or incoherent action sequences. These results unequivocally show that C_F and C_Q play distinct, non-redundant roles, and that our decoupled design is key to resolving Signal Confounding.

Table 3: Ablation study results on the VirtualHome benchmark. We compare the full DDCG framework against variants without the Feasibility Critic (w/o C_F) and without the Quality Critic (w/o C_Q).

Setting	**Model**	**Exec. (%)**	**SR (%)**
In-Distribution	DDCG (Ours)	**95.0**	**94.1**
	w/o C_F	89.4	82.3
	w/o C_Q	93.1	90.0
Novel Scenes	DDCG (Ours)	**75.6**	**73.6**
	w/o C_F	74.2	70.1
	w/o C_Q	70.5	63.2
Novel Tasks	DDCG (Ours)	**83.3**	**81.8**
	w/o C_F	78.9	75.0
	w/o C_Q	82.5	80.0

5 Conclusion

In this work, we identified "Signal Confounding"—the ambiguity in feedback signals that plagues current LLM-based planners—as a critical barrier to creating robust and efficient embodied agents. We introduced DDCG, a novel framework that directly addresses this problem by decoupling the decision-making process into two distinct, specialized modules: a Feasibility Critic (C_F) that enforces environmental rules and a Quality Critic (C_Q) that guides strategic choice. Our experimental results show that by separating the question of "Can I do this?" from "Should I do this?", DDCG achieves superior performance in both task success and planning efficiency.

References

[1] Eitan Altman. *Constrained Markov decision processes*. Routledge, 2021.

[2] Chenjia Bai, Huazhe Xu, and Xuelong Li. Embodied-ai with large models: research and challenges. *SCIENTIA SINICA Informationis*, 54(9), 2024.

[3] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology*, 2024.

[4] Zhenfang Chen, Delin Chen, Rui Sun, Wenjun Liu, and Chuang Gan. Autonomous agents from automatic reward modeling and planning. In *The Thirteenth International Conference on Learning Representations*, 2025.

[5] Xian Fu, Min Zhang, Jianye Hao, Peilong Han, Hao Zhang, Lei Shi, and Hongyao Tang. What can vlms do for zero-shot embodied task planning? *Workshop on Large Language Models and Cognition at the 41st International Conference on Machine Learning*, 2024.

[6] Yixuan Guo, Yuda Zhao, Zhidian Zhang, Zike Huang, Jian Gao, and Jian Gao. Discriminator-guided embodied planning for llm agent. *arXiv preprint arXiv:2310.09348*, 2023.

[7] Mengkang Hu, Yao Mu, Xinmiao Yu, Mingyu Ding, Shiguang Wu, Wenqi Shao, Qiguang Chen, Bin Wang, Yu Qiao, and Ping Luo. Tree-planner: Efficient close-loop task planning with large language models.

[8] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, pages 9118–9140. PMLR, 2022.

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

[10] Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. Pre-trained language models for interactive decision-making. *arXiv preprint arXiv:2202.01771*, 2022.

[11] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36, 2024.

[12] OpenAI. Gpt-4 technical report, 2023.

[13] Xavier Puig, Kevin Ra, Marko Teng, Tianmin Li, Sabine Süsstrunk, and Alan Jacobson. Virtualhome: A virtual environment for instruction giving and execution. In *CVPR Workshops*, page 25, 2018.

[14] Yuxiao Qu, Tianjun Zhang, Naman Garg, and Aviral Kumar. Recursive introspection: Teaching language model agents how to self-improve, 2024.

[15] William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. Self-critiquing models for assisting human evaluators, 2022.

[16] Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *arXiv preprint arXiv:2303.11366*, 2023.

[17] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11499–11505, 2023.

[18] Jerry Wei, Chengrun Yang, Xinying Song, Yifeng Lu, Nathan Hu, Jie Huang, Dustin Tran, Daiyi Peng, Ruibo Liu, Da Huang, Cosmo Du, and Quoc V. Le. Long-form factuality in large language models, 2024.

[19] Yike Wu, Jiatao Zhang, Nan Hu, Lanling Tang, Guilin Qi, Jun Shao, Jie Ren, and Wei Song. MLDT: multi-level decomposition for complex long-horizon robotic task planning with open-source large language model. In *DASFAA (5)*, volume 14854 of *Lecture Notes in Computer Science*, pages 251–267. Springer, 2024.

[20] Yu Xia, Jingru Fan, Weize Chen, Siyu Yan, Xin Cong, Zhong Zhang, Yaxi Lu, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Agentrm: Enhancing agent generalization with reward modeling.

[21] Ruihan Yang, Fanghua Ye, Jian Li, Zhaopeng Tu, Xiaolong Li, Siyu Yuan, Yikai Zhang, and Deqing Yang. The lighthouse of language: Enhancing llm agents via critique-guided improvement. *arXiv preprint arXiv:2503.16024*, 2025.

[22] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

[23] Dongqi Zuo, CHEN Zheng, Chuan Zhou, Yandong Guo, Xiao He, and Mingming Gong. Radi: Llms as world models for robotic action decomposition and imagination. In *ICLR 2025 Workshop on World Models: Understanding, Modelling and Scaling*.

A Appendix

A.1 Prompt Templates

To elicit plans from the LLM, we designed and tested several prompt templates. The main templates used in our study are detailed below.

General Task Planning. This template is a monolithic prompt that provides the LLM with all necessary context in a single block to directly generate a full sequence of actions. The structure includes sections for predefined rules, the initial state of the environment, a list of available actions, the specific task instruction, and finally a section for the LLM to generate its plan.

```
# Long-term Memory
# Action Primitives:
from actions import walk <obj>, grab <obj>, switchon <obj>, switchoff <obj>, open <obj>,
close <obj>, turnto <obj>, drink <obj>, putin <obj> <obj>, putback <obj> <obj>

# Rules:
# remeber if the key object INSIDE kitchencabinet, you should open the kitchencabinet first or
the key object INSIDE room, you should walk to the room
# and different id represent different items, so note the id number.
# remeber you should grab only one item at a time and you can not open a cabinet that has
been opened

# Overall Task Goal:
{total_task_goal}

# Completed Sub-goals:
{completed_sub_goals}

# Recently Executed Actions (Optional, with scores):
{recent_actions_with_scores}

# Short-term Memory
# Few-shot Examples:
{few_shot_examples}

# Current Environment State:
# remember the key object locations and states: {current_env_state}

# Current Task
# The task goal: {current_sub_goal}
```

Figure 2: The General Task Planning prompt template. It combines rules, examples, and the current state to prompt the LLM for a complete action sequence in one pass.

Reasoner + Planner. This template employs a hierarchical, two-step approach. First, a 'Reasoner' component is prompted to decompose the high-level task into a sequence of more abstract subgoals. Subsequently, a 'Planner' component is prompted with a specific subgoal from the sequence to generate the corresponding low-level, executable actions required to complete that step.

Replanning. This template is designed for error correction and uses a history of failed attempts as explicit negative feedback. The prompt contains a dedicated section for previously unsuccessful plans. The LLM is then prompted to generate a new plan with the explicit instruction to avoid repeating the previous mistakes, thereby guiding the model toward a valid solution.

A.2 Limitations

The performance of our DDCG framework is dependent on the quality of the synthetically generated data and introduces computational overhead, while its generalization to physical robotics requires further validation.

```
# Now you are a task planning assistant, responsible for inferring the execution steps of a task.
# You should mimic the provided examples and, based on the task objectives, understand the
total task goal first, generate the next sub-task.

# There are some examples:
# {reasoner_few_shot_examples}

# Imitate these examples to generate a step-by-step plan.
# Task goal: {task_goal
```

```
# Now you are a task planning assistant. You should mimic the examples I provide and
generate a sequence of actions based on the target instructions and environmental
information. Pay attention to the task objectives and environmental information.

# And remember if the key object INSIDE kitchencabinet, you should open the kitchencabinet
first... (Rules from planner_rule)

# There are some examples:
{planner_few_shot_examples}

# Imitate these examples to generate an action list.

# Now the task is: {reasoned_sub_task}
# remember the key object locations and states: {current_env_state}
```

Figure 3: The Reasoner + Planner prompt template. This hierarchical prompt first generates high-level steps and then plans low-level actions for each step.

```
# You are a task planning assistant. Your goal is to complete the task: '{total_task_goal}'.
The current sub-goal is: '{current_sub_goal}'.

# Here are the only actions you are allowed to use:
walk <obj>, grab <obj>, switchon <obj>, switchoff <obj>, open <obj>, close <obj>, turnto
<obj>, drink <obj>, putin <obj> <obj>, putback <obj> <obj>

Here are some important rules:
# remember you should grab only one item at a time.
# you can not open a cabinet that has been opened.
# if the key object is INSIDE another object, you should open it first.

# Here is the history of recent actions and why they were bad:
{failed_actions_history}

# Based on this information, please provide a better, single next action to take FROM THE
ALLOWED LIST.
# Do not repeat the failed actions.
# Just provide the single action command in the format like 'grab('kitchenknife(id:123)')'.
```

Figure 4: The Replanning prompt template. It uses a history of failed actions as negative feedback to guide the LLM in generating a corrected plan.