

CARL: Constraint-Aware Reinforcement Learning for Planning with LLMs

Qiuyi Qi^{♠♦*}, Jinjian Zhang^{♦*}, Mutian Bao^{♠♦*}, Tian Liang^{♠♦}, Guocong Li^{♠♦},
Dongnan Liu[♦], Wei Zhou[♦], Jie Liu[♠], Ming Kong^{♠†},
Linjian Mo^{♦†}, Feng Zhang[♠], Qiang Zhu^{♠†}
♠ Zhejiang University, ♦ Ant Group, ♠ City University of Hong Kong
{qiqiuyi, zhuq}@zju.edu.cn

Abstract

Despite their strong reasoning capabilities and extensive world knowledge, Large Language Models (LLMs) frequently generate plans that violate task constraints, undermining their reliability in real-world applications. This deficiency arises from a lack of systematic mechanisms to incorporate constraint information during the generation process. While existing approaches attempt to mitigate this by relying on external tools or task decomposition, they fail to enhance the model’s intrinsic constraint awareness. To address this, we propose Constraint-Aware Reinforcement Learning (CARL), a novel RL framework designed to strengthen LLMs’ intrinsic focus on constraints. CARL introduces a constraint-aware reward by comparing the model’s output distributions under constrained and unconstrained inputs, encouraging constraint focus and penalizing neglect. Compatible with various RL frameworks and requiring no external solvers or top models, CARL enables scalable, end-to-end constraint-aware planning. Extensive experiments on BlocksWorld, TravelPlanner, and T-Eval demonstrate that CARL significantly outperforms standard Reinforcement Fine-Tuning (RFT) baselines and state-of-the-art reasoning models, exhibiting a markedly increased focus on constraints.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable capabilities in reasoning, tool utilization, and world knowledge modeling, positioning them as powerful candidates for complex planning tasks—a cornerstone of cognitive AI systems (Huang et al., 2022b; Ahn et al., 2022). Plan-

*Q. Qi, J. Zhang and M. Bao contributed equally to this work.

†Q. Zhu, M. Kong and L. Mo are corresponding authors. Q. Zhu is with the College of Artificial Intelligence, Shanghai Institute for Advanced Study, Zhejiang University. M. Kong is with the School of Earth Sciences, Zhejiang University. L. Mo is with the Ant Group.

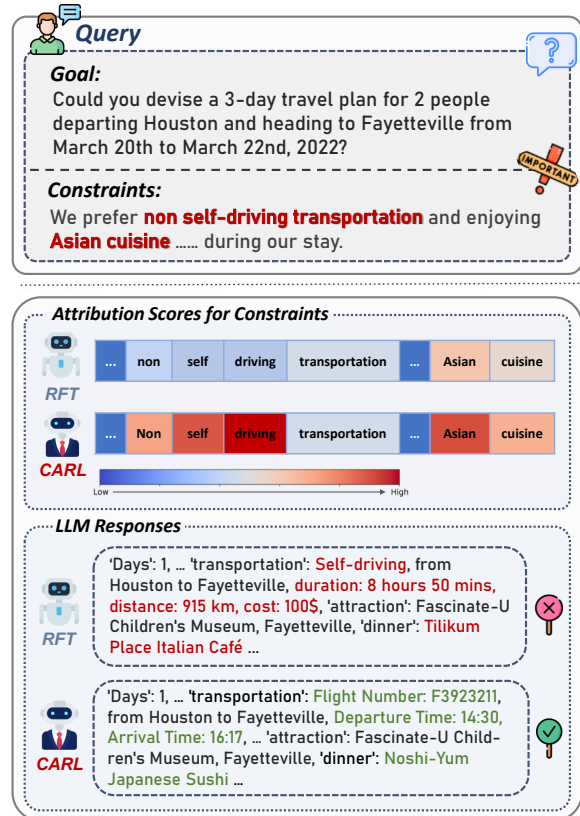


Figure 1: A typical case of planning tasks. The upper box demonstrates that a query can be decomposed into a goal and constraints, while the lower box shows that our CARL exhibits a higher focus on constraints, ultimately outperforming RFT in planning.

ning entails generating a sequence of executable actions to achieve a goal while strictly adhering to a set of constraints (Newell et al., 1958; Kartam and Wilkins, 1990). For instance, a travel itinerary must not only satisfy the destination and timeline but also comply with specific constraints such as *budget limits*, *transportation preferences*, or *dietary requirements*.

Despite these capabilities, LLMs consistently struggle to generate constraint-compliant plans in practice (Wei et al., 2025; Huang et al., 2024). On

the challenging real-world benchmark TravelPlanner (Xie et al., 2024a), DeepSeek-R1 (Guo et al., 2025), a model renowned for its general reasoning performance, achieves a pass rate of only 12.2%, falling significantly short of human-level performance. This disparity is not due to weak reasoning ability but rather reflects a fundamental limitation: **LLMs lack the capacity to systematically incorporate constraints into their generation process.** Empirical studies (Xie et al., 2024b) corroborate this, revealing that LLMs frequently neglect constraints during planning and exhibit low attribution scores for constraint-related tokens in the input.

Existing approaches primarily circumvent this issue by offloading constraint reasoning to external scaffolds. Common paradigms include *plan-then-execute*, which decomposes complex queries into a sequence of simpler subtasks (Wang et al., 2023; Singh et al., 2022); *step-by-step* frameworks that interleave planning with action execution in an iterative manner (Wei et al., 2022); and neuro-symbolic methods that translate natural language queries into formal planning representations (e.g., PDDL) for symbolic solvers (Wu et al., 2022; He-Yueya et al., 2023). While these methods yield performance gains, they rely heavily on closed-source models, external tools, or task-specific engineering. Crucially, they do not enhance the model’s intrinsic understanding of constraints, thereby limiting generalizability and deployment in autonomous environments.

To address this, we propose **Constraint-Aware Reinforcement Learning (CARL)**, a general RL framework designed to explicitly strengthen LLMs’ focus on constraints. CARL introduces a novel constraint-aware reward derived from the distributional shift of the model’s outputs under constrained versus unconstrained inputs. By utilizing the KL divergence between log-probabilities in these two settings as a reward signal, CARL guides the model to integrate constraint signals more effectively. Unlike discrete task rewards, our continuous constraint-aware reward provides smoother optimization gradients and encourages meaningful exploration during planning failures.

CARL provides a novel and effective training framework for enhancing intrinsic planning competence. As illustrated in Figure 1, models trained with CARL exhibit significantly higher attribution scores for constraint tokens (e.g., non-self-driving transportation, Asian cuisine) compared to standard RFT, leading to improved plan validity. Further-

more, CARL is model-agnostic and compatible with a broad spectrum of RL algorithms, including on-policy methods like PPO (Yu et al., 2022) and GRPO (Shao et al., 2024), as well as off-policy approaches like DPO (Rafailov et al., 2023).

We evaluate CARL across three diverse planning benchmarks: BlocksWorld (block manipulation) (Valmeekam et al., 2024), TravelPlanner (travel planning) (Xie et al., 2024a), and T-Eval (tool use) (Chen et al., 2023). The results demonstrate consistent and substantial improvements over RFT. Notably, CARL achieves a 56.1% final pass rate on TravelPlanner—a +11.1% gain over the baseline—outperforming state-of-the-art reasoning models such as o1-preview (10.0%) and DeepSeek-R1 (12.2%). Ablation studies and attribution analysis further confirm that these gains stem from a successfully acquired awareness of constraints. Our contributions are summarized as follows:

- We propose CARL, a novel RL framework that systematically enhances LLMs’ focus on constraints by modeling distributional shifts in output log-probabilities under constrained versus unconstrained inputs.
- We design a learnable and interpretable constraint-aware reward mechanism that enables fine-grained control over constraint compliance. This formulation is highly generalizable, seamlessly extending to both on-policy and off-policy paradigms.
- We demonstrate CARL’s effectiveness across diverse planning benchmarks, where it achieves substantial performance gains and significantly improved constraint focus, all without reliance on external solvers or top models.

2 Related Works

2.1 Planning with LLMs

Large Language Models (LLMs) have exhibited impressive potential in complex planning tasks, driven by their reasoning capabilities (Yao et al., 2023; Kojima et al., 2022; Raman et al., 2024) and proficiency in tool utilization (Qin et al., 2023; Schick et al., 2023). Leveraging LLMs’ zero-shot generalization, early studies explored direct planning approaches (Huang et al., 2022a; Ahn et al., 2022), though these were often limited to simple, grounded scenarios. To tackle more complex

problems, Chain-of-Thought (CoT) prompting was adopted to induce structured reasoning (Wei et al., 2022). More recently, tool-augmented frameworks have emerged to bolster planning reliability: some translate problems into formal logic for external solvers (Liu et al., 2023; Xie et al., 2023; Gundawar et al., 2024), others integrate code snippets to handle dynamic "what-if" scenarios (Li et al., 2023), and several implement iterative refinement loops utilizing task-specific verifiers or human feedback (Kambhampati et al., 2024; Chen et al., 2024).

A common limitation of these approaches is that they treat LLMs as static components—either as high-level dispatchers or translators—rather than fundamentally improving their planning proficiency. Consequently, they externalize complex reasoning and constraint handling, creating dependencies on external tools and top models. While fine-tuning offers a direct pathway to acquire intrinsic skills, its application to planning tasks remains under-explored, particularly regarding the systematic integration of constraints during generation.

In contrast, we propose a novel RL paradigm that directly enhances the model’s constraint-aware planning capabilities at the policy level. Instead of relying on external scaffolding, our method instills planning competence directly into the model via the constraint-aware reward, enabling superior performance and autonomy without dependence on external solvers, top models, or extensive prompt engineering.

2.2 Reinforcement Learning for LLMs

Reinforcement Learning from Human Feedback (RLHF) (Kaufmann et al., 2024) adapts RL for LLMs, traditionally employing Proximal Policy Optimization (PPO) (Yu et al., 2022). To improve stability and efficiency, methods like Direct Preference Optimization (DPO) (Rafailov et al., 2023) and SimPO (Meng et al., 2024) have been developed, though often at the cost of on-policy performance. Recent advances such as Group Relative Policy Optimization (GRPO) (Shao et al., 2024) and Reinforcement Learning with Online Optimization (RLOO) (Ahmadian et al., 2024) seek to balance performance with computational efficiency.

Parallel research has attempted to enhance input awareness through specialized rewards or preference pairs. For example, Kiruluta et al. (2025) utilize attention scores as reward signals to prioritize crucial tokens, while Gu et al. (2024); Deng et al. (2024) construct preference pairs from original and

noise-perturbed images to strengthen visual anchoring. However, these methods are primarily limited to simple QA or visual tasks, and their efficacy in complex reasoning scenarios such as planning remains unverified.

In this work, we introduce a constraint-aware reward mechanism tailored for planning tasks. Our key innovation lies in quantifying constraint sensitivity through the distributional shift in log-probabilities between constrained and unconstrained conditions. This formulation is natively compatible with on-policy methods (PPO, GRPO) and can be seamlessly adapted to off-policy frameworks like DPO by transforming constraint signals into preference pairs, ensuring robust constraint compliance across training paradigms.

3 Preliminaries

3.1 Planning Task Formulation and Decomposition

To faithfully address the challenges in planning tasks, as shown in Figure 2, a planning task query can be decomposed into two components: goal (the final target to achieve) and constraints (the conditions that need to be adhered to). On this basis, we define an unconstrained planning problem to more systematically examine the role of constraints in achieving reliable planning outcomes. Our method is designed to be broadly applicable across generic planning scenarios. Specifically, a planning task \mathcal{P} is represented as an input sequence:

$$x = (x_1, x_2, \dots, x_T) \in \mathbb{R}^{T \times d} \quad (1)$$

where T is the input length, and d is the dimensionality of the embedding space.

Constraints are an inherent part of all planning tasks and play a pivotal role in ensuring outcome correctness. In our framework, such constraints are identified and modeled through the following procedures:

- For tasks with explicitly stated constraints, such as TravelPlanner, we utilize the predefined constraints explicitly described within the task query.
- For tasks with implicitly defined constraints, such as T-Eval, we employ constraint-extraction heuristics. Specifically, a lightweight prompt-based approach extracts and isolates the constraint-relevant portions

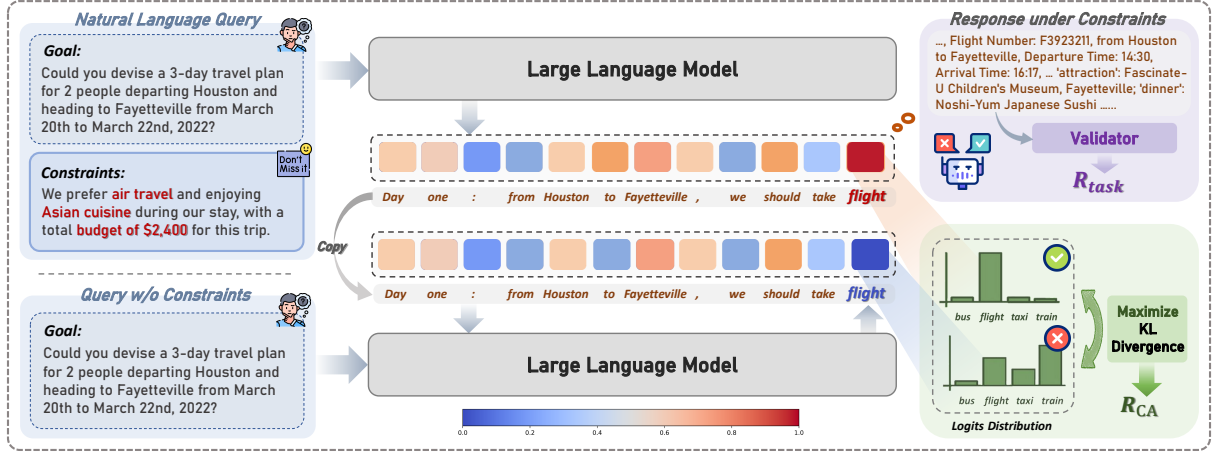


Figure 2: The framework of our proposed CARL. The reward is decomposed into two components: R_{task} for achieving task-oriented objectives and R_{CA} for sensitive adherence to constraints.

of the input query, with full implementation details and sensitivity analysis provided in Appendix A.

Formally, we denote the indices corresponding to the constraint tokens as $\mathcal{C} \subseteq \{1, 2, \dots, T\}$. The extracted sequence of constraint-specific tokens is then $x_{\mathcal{C}} = (x_t)_{t \in \mathcal{C}}$. The remaining subsequence, which represents the unconstrained portion of the planning task, is defined as:

$$x_{\setminus \mathcal{C}} = (x_t)_{t \in \{1, \dots, T\} \setminus \mathcal{C}} \quad (2)$$

While the unconstrained planning task $\mathcal{P}_{\setminus \mathcal{C}}$ may capture general goal-related elements of the task, it omits the crucial information encoded within constraints, potentially leading to incomplete or invalid solutions. This observation motivates our decomposition approach: constraints often act as the governing principles that disambiguate solutions and guarantee their feasibility. This distinction sets the stage for our proposed constraint-aware reinforcement learning framework.

3.2 Group Relative Policy Optimization (GRPO)

GRPO (Shao et al., 2024) is an on-policy reinforcement learning algorithm. In the context of planning, consider a dataset D containing datapoints consisting of inputs x . The GRPO learning objective with respect to the policy π_{θ} can be written as follows, where θ represents the parameters in a large lan-

guage model:

$$\begin{aligned} \mathcal{J}_{\text{GRPO}}(\theta) = & \mathbb{E}_{\{y_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(Y|x)} \frac{1}{G} \sum_{i=1}^G \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \left\{ \right. \\ & \min \left[r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1 - \epsilon_l, 1 + \epsilon_h) \hat{A}_{i,t} \right] \\ & \left. - \beta \mathbb{D}_{\text{KL}}[\pi_{\theta} \parallel \pi_{\text{ref}}] \right\} \\ \text{with } r_{i,t}(\theta) = & \frac{\pi_{\theta}(y_{i,t}|x, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|x, y_{i,<t})} \end{aligned} \quad (3)$$

G denotes the size of the group which contains multiple responses Y sampled from the rollout policy $\pi_{\theta_{\text{old}}}$, corresponding to one input instance x . $\epsilon_l, \epsilon_h \in \mathbb{R}$ are hyperparameters for clipping too large updates. The token-level advantage $\hat{A}_{i,t}$ is defined as the sequence-level reward normalized across the group.

4 Methodology

4.1 Overview

To address the dual objectives of goal attainment and constraint satisfaction in planning, we reformulate the standard reinforcement learning (RL) objective by decomposing the overall reward signal R into two components, as shown in Figure 2: a task-specific reward R_{task} for achieving task-oriented objectives, and a constraint-aware reward R_{CA} for sensitive adherence to constraints:

$$R(x, y) = R_{\text{task}}(x, y) + \alpha R_{CA}(x, y) \quad (4)$$

where $\alpha \geq 0$ is a hyperparameter that regulates the relative importance of constraint adherence in the learning process. The trade-off between these components balances task achievement with constraint compliance.

At its core, our methodology embeds constraint-awareness into both the reward shaping and the policy optimization steps, detailed below.

4.2 Constraint-Aware Reward Shaping

We introduce a novel reward shaping mechanism that explicitly integrates constraint-awareness into the learning objective. The task reward R_{task} is computed as:

$$R_{\text{task}}(x, y) = \mathbb{I}[y \in \mathcal{Y}_{\text{valid}}(x)] \quad (5)$$

where $\mathbb{I}[\cdot]$ is an indicator evaluating to 1 if the generated output y satisfies the task-specific success criterion (task achievement), and $\mathcal{Y}_{\text{valid}}(x)$ is the task-valid output space for input x . The constraint-aware reward R_{CA} is defined as:

$$R_{\text{CA}}(x, y) = \mathbb{D}_{\text{KL}}[\pi_{\theta}(y|x) \parallel \pi_{\theta}(y|x_{\setminus \mathcal{C}})] \quad (6)$$

Intuitively, the KL divergence captures the extent to which constraints influence model behavior, encouraging generation patterns that adhere to the constraint information. Combining this reward with the RL objective (e.g., GRPO) yields the complete CARL objective:

$$\begin{aligned} \mathcal{J}_{\text{CARL}}(\theta) = & \mathbb{E}_{\{y_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(Y|x)} \frac{1}{G} \sum_{i=1}^G \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \left\{ \right. \\ & \min \left[r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip}(r_{i,t}(\theta), 1 - \epsilon_l, 1 + \epsilon_h) \hat{A}_{i,t} \right] \\ & - \beta \mathbb{D}_{\text{KL}}[\pi_{\theta} \parallel \pi_{\text{ref}}] \\ & \left. + \alpha \mathbb{D}_{\text{KL}}[\pi_{\theta}(y_i|x) \parallel \pi_{\theta}(y_i|x_{\setminus \mathcal{C}})] \right\} \end{aligned} \quad (7)$$

where i indexes the i -th rollout response. α and β are weighting coefficients used for constraint-aware reward and KL penalty ($\mathbb{D}_{\text{KL}}[\pi_{\theta} \parallel \pi_{\text{ref}}]$). We then compare the GRPO-version training dynamics and efficiency of CARL and RFT, as shown in Appendix B.

4.3 Constraint-Aware Direct Preference Optimization

To adapt our constraint-aware paradigm to preference-based learning, we propose a novel extension of Direct Preference Optimization (DPO) that injects constraint focus through strategic preference construction. Our key innovation lies in generating contrastive responses under constraint ablation to create informative preference pairs.

Given a standard response $y \sim \pi(\cdot|x)$ and its constraint-ablated counterpart $y_{\setminus \mathcal{C}} \sim \pi(\cdot|x_{\setminus \mathcal{C}})$, the

preference dataset is defined as:

$$\mathcal{D}_{\text{CA}} = \left\{ (x^{(i)}, y^{(i)}, y_{\setminus \mathcal{C}}^{(i)}) \mid y^{(i)} \in \mathcal{Y}_{\text{valid}}(x^{(i)}) \right\}_{i=1}^N \quad (8)$$

The preference-based DPO objective is then augmented to reflect the impact of constraint ablation:

$$\mathcal{L}_{\text{CA-DPO}} = -\mathbb{E}_{\mathcal{D}_{\text{CA}}} [\log \sigma(\beta \Delta(x, y, y_{\setminus \mathcal{C}}))] \quad (9)$$

The constrained advantage function Δ is computed as:

$$\Delta(x, y, y_{\setminus \mathcal{C}}) = \log \frac{\pi_{\theta}(y|x)}{\pi_{\text{ref}}(y|x)} - \log \frac{\pi_{\theta}(y_{\setminus \mathcal{C}}|x)}{\pi_{\text{ref}}(y_{\setminus \mathcal{C}}|x)} \quad (10)$$

This formulation introduces an implicit reward margin that quantifies constraint influence on policy outputs, effectively incentivizing the model to generate constraint-compliant responses through optimization. The corresponding results are shown in Figure 3.

5 Experiments

5.1 Datasets and Settings

Datasets. We evaluate CARL across three complementary planning benchmarks that collectively cover *classical symbolic planning*, *real-world constrained decision-making*, and *tool-mediated planning*:

- **BlocksWorld** (Valmeekam et al., 2024) is a formal symbolic planning environment with well-defined action schemas and **static** constraints. Given an initial block configuration and a goal state, models must generate action sequences that strictly adhere to the physical constraints specified in the prompt. This benchmark provides a controlled setting to isolate and evaluate core constraint-handling capabilities.
- **TravelPlanner** (Xie et al., 2024a) presents a real-world travel planning challenge where models must generate plans based on provided information and user queries, aligning with commonsense and the hard constraints specified in the queries. Unlike the static nature of BlocksWorld, the hard constraints in TravelPlanner are **dynamic**, as they need to be inferred from the query and satisfied through item selection.

Model	BlocksWorld	TravelPlanner					T-Eval			
		Commonsense		Hard		Final	Precision	Recall	F1-score	
		Micro	Macro	Micro	Macro					
GPT-4o	42.4	84.7	31.1	53.6	31.1	7.8	90.4	86.4	87.5	
o1-preview	97.8	79.6	15.0	41.9	37.8	10.0	90.0	86.5	87.4	
DeepSeek-V3	44.8	80.3	17.2	30.5	13.9	2.2	91.1	87.4	88.5	
DeepSeek-R1	98.2	80.6	22.2	51.7	41.7	12.2	90.2	87.2	87.8	
Qwen2.5-72B-Instruct	13.8	82.3	16.7	32.6	22.8	6.1	92.2	88.1	89.2	
QwQ-32B	88.8	74.9	6.1	41.4	32.8	4.4	88.3	84.6	85.6	
Llama-3.1-70B-Instruct	21.6	82.8	18.9	33.1	16.1	2.2	85.4	81.9	83.0	
Llama-3.1-8B-Instruct	0.6	60.1	0.0	7.9	2.8	0.0	81.5	76.5	78.9	
DeepSeek-R1-Distill-Llama-8B	1.4	61.2	0.0	0.0	0.0	0.0	81.8	79.3	79.4	
Qwen3-8B	31.2	72.7	7.8	34.8	27.8	2.2	86.6	83.5	84.2	
DeepSeek-R1-Distill-Llama-8B (RFT)	42.0	80.8	25.0	36.2	19.4	5.6	88.0	86.0	86.4	
DeepSeek-R1-Distill-Llama-8B (Ours)	52.6	81.1	32.1	42.9	28.9	11.7	88.4	88.6	<u>87.5</u>	
Qwen3-8B (RFT)	<u>73.8</u>	<u>96.3</u>	<u>74.4</u>	<u>65.7</u>	<u>48.9</u>	<u>45.0</u>	<u>88.6</u>	88.2	87.3	
Qwen3-8B (Ours)	77.2	97.3	81.7	73.1	59.4	56.1	89.5	<u>88.5</u>	88.1	

Table 1: Results on planning benchmarks. Unless otherwise specified, both RFT and CARL are implemented based on GRPO. The best and second-best results are **bold** and underlined.

- **T-Eval** (Chen et al., 2023) is a fine-grained benchmark assessing LLMs’ tool-use ability across multiple evaluation aspects. In this work, we primarily focus on its planning task. As noted in Sec. 3.1, T-Eval is characterized by **implicit** constraints, which are embedded within the input queries. We then use GPT-4o with a lightweight prompt to extract these constraints.

We follow the official partitions for BlocksWorld and TravelPlanner, using 100 and 45 samples for training, and 500 and 180 samples for testing, respectively. For T-Eval, we randomly select 128 samples from the 553-sample evaluation set for training, with the remaining samples for testing.

Metrics. Accuracy is used for evaluating BlocksWorld, whereas precision, recall, and F1-score are used for T-Eval¹. For TravelPlanner, we employ a multi-faceted evaluation framework that separates commonsense from hard constraints, with two complementary metrics reported for each:

- *Micro pass rate*: The ratio of successfully satisfied constraints to total constraints of that type.
- *Macro pass rate*: The ratio of plans satisfying all constraints of that type to total plans.

Finally, we use the *final pass rate* as the proportion of plans satisfying all constraints, which

¹In T-Eval, the reported precision, recall, and F1-score are the arithmetic means of the per-sample scores.

corresponds to the *macro pass rate* when considering all constraints collectively, representing the ultimate planning success metric.

Implementation Details. Our reinforcement learning framework is implemented based on Verl (Sheng et al., 2025). Unless otherwise specified, we adopt Qwen3-8B as the base model to balance performance and training efficiency, and use GRPO for optimization. Training is conducted on a single node with 8 A100 GPUs, and each step samples a batch of 64 queries with 8 rollouts. The weighting coefficients α and β (see Equation 7) are both set to 0.001. Additional training details are provided in Appendix C.

5.2 CARL Performance

Results on Planning Benchmarks. Table 1 summarizes the main evaluation results across all planning benchmarks, highlighting the comparative performance of CARL against state-of-the-art LLMs and RFT. There are three key takeaways:

First, CARL significantly boosts the planning capabilities of Qwen3-8B and DeepSeek-R1-Distill-Llama-8B. For example, with fewer than 128 training queries, CARL significantly boosts Qwen3-8B’s planning performance, achieving an absolute improvement of 53.9% on TravelPlanner (from 2.2% to 56.1%). It consistently outperforms RFT and significantly surpasses state-of-the-art reasoning-oriented models such as o1-preview (10.0%) and DeepSeek-R1 (12.2%). Furthermore, CARL enables the performance of the 14B model comparable to top models on BlocksWorld, as

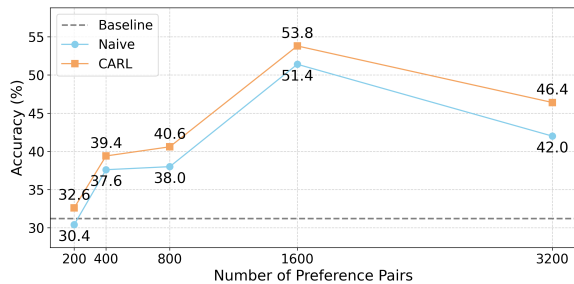


Figure 3: Performance comparison of DPO and CA-DPO with different preference pairs on BlocksWorld.

shown in Figure 4 and Table 11.

Second, CARL effectively mitigates constraint neglects or violations in multi-constraint planning tasks. While baseline models often perform well in terms of *micro pass rate* (i.e., the proportion of individual constraints satisfied), their *macro pass rate* (i.e., the proportion of plans satisfying all constraints) remains low. This indicates that performance bottlenecks arise from constraint neglect or violation in multi-constraint scenarios, rather than inherent difficulty in satisfying individual constraints. By enhancing the model’s focus on constraints, CARL substantially mitigates such issues, leading to improved *macro* and *final pass rates*.

Last, CARL outperforms RFT in handling dynamic hard constraints. Although both CARL and RFT support the learning of static commonsense constraints, CARL proves more effective at handling dynamic constraints—hard constraints that vary across instances—resulting in higher overall task success rates than RFT.

To summarize, CARL consistently demonstrates superior planning performance and robustness across benchmarks and models. Additional experimental results, including comparisons with SFT, prompting, and agent-based methods, as well as generalization performance on other benchmarks, are presented in Appendix D.

Adaptation to Other RL Methods. Our proposed CARL framework can be seamlessly extended to other representative on-policy and off-policy reinforcement learning methods, such as PPO and DPO. Table 2 summarizes the performance of these CARL implementations, consistently showing superior results compared to their naive counterparts.

The implementation of CARL on PPO is similar to that on GRPO; however, for DPO, we begin by constructing preference pairs using responses

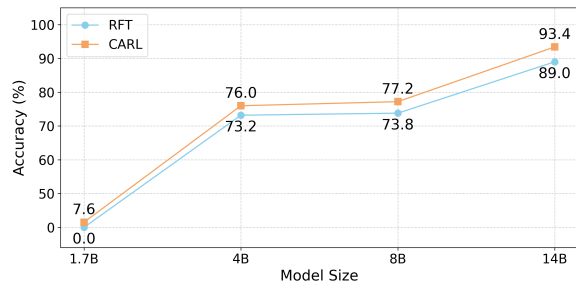


Figure 4: Performance comparison of RFT and CARL with different size Qwen3 models on BlocksWorld.

Model	PPO		DPO	
	Naive	CARL	Naive	CARL
Qwen3-8B	78.4	81.6	51.4	53.8

Table 2: Performance comparison of RFT and CARL with different RL methods on BlocksWorld.

generated from the base model. Specifically, we input constrained queries and construct preference pairs based on the correctness of the responses for naive DPO. For CA-DPO, we input both constrained and unconstrained queries. The correct responses generated for constrained queries are selected as positive samples, while the responses generated for unconstrained queries are randomly chosen as negative samples, which are then utilized to construct preference pairs. For both methods, we maintain the same total number of preference pairs and use identical parameter settings for optimization. Experimental results, shown in Figure 3, reveal that CA-DPO consistently outperforms naive DPO. With as few as 200 preference pairs, CARL enables the model to be aware of constraints and slightly improves planning performance. As the number of preference pairs increases, the diversity of data expands, leading to a continuous improvement in the model’s planning capabilities. However, when the number of preference pairs reaches 3,200, the model’s performance begins to decline. We attribute this to a bottleneck in data diversity among the preference pairs generated by the base model. We think that introducing additional data sources in future work could further enhance the effectiveness of DPO. For more details about DPO, please refer to Appendix E.

5.3 Ablation Studies

We demonstrate the effectiveness of our constraint-aware reward through ablation studies of model

Model	None	Both	Goal	Constraint
Qwen3-8B	73.8	71.2	70.4	77.2

Table 3: Ablation study of reward masking strategy on BlocksWorld. “Both” denotes masking both goals and constraints.

size, reward masking, and reward calculation strategies. Additionally, we investigate the sensitivity of the constraint-aware reward weight (i.e., hyperparameter α) in Appendix F.

Model Size. To investigate the impact of model scale, we apply CARL to four Qwen3 models of increasing size. Results in Figure 4 highlight two key effects:

- **Exploration Guidance:** For small models lacking initial planning ability (e.g., Qwen3-1.7B), RFT fails due to the absence of task rewards during rollout. In contrast, CARL’s constraint-aware reward provides smoother gradients, enabling effective exploration and gradual acquisition of planning skills through constraint focus.
- **Constraint Grounding:** For models with basic planning ability, CARL’s advantage over RFT grows with scale, as larger models better leverage constraint signals to activate stronger reasoning patterns and boost planning performance.

Reward Masking Strategy. We assess the reward design by extending masking targets to goals or both components. As shown in Table 3, these variants perform worse than RFT (None). We hypothesize that removing goals disrupts the core task definition, leading to incoherent outputs and noisy supervision. In contrast, queries without constraints can still produce valid plans, allowing the reward to suppress constraint neglect.

Reward Calculation Strategy. To investigate the impact of the reward calculation strategy on training performance and dynamics, we measure the discrepancy between the log-probabilities in Equation 6 using four metrics: *difference* (dif), *absolute difference* (abs), *mean squared error* (mse), and *low-variance kl divergence* (low_var_kl, which we used in our other experiments). As shown in Table 4, all strategies facilitate stable training, with

Model	dif	abs	mse	low_var_kl
Qwen3-8B	76.6	76.0	0.2	77.2

Table 4: Ablation study of reward calculation strategy on BlocksWorld. “dif”, “abs”, “mse”, and “low_var_kl” denote different discrepancy metrics.

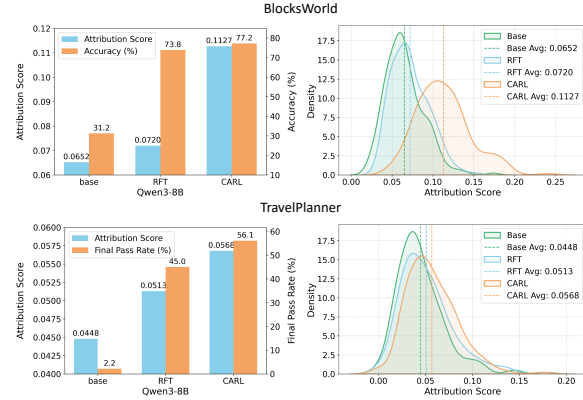


Figure 5: Attribution analysis on BlocksWorld and TravelPlanner. The left simultaneously presents the average attribution scores and performance of different models, while the right illustrates score distributions through Kernel Density Estimation.

the notable exception of mse, which leads to collapse. The impact of reward calculation strategy on training dynamics is shown in Appendix G.

5.4 Attribution Analysis

To understand the source of planning performance improvements, we conduct an attribution analysis on BlocksWorld and TravelPlanner. We compare the base model with versions fine-tuned using RFT and CARL, focusing on the mean and distribution of attribution scores for constraint-related inputs (all scores are normalized by the response length). As shown in Figure 5, better task performance consistently correlates with higher attribution scores. Compared to RFT, CARL further enhances the model’s focus on constraints, leading to stronger performance. Importantly, this improvement is not driven by a few outliers but reflects a consistent shift in the overall distribution, indicating that CARL’s gains stem from a general enhancement in constraint sensitivity. Additional details and the case study are provided in Appendix H.

6 Conclusion

In this paper, we introduced CARL, a novel RL framework designed to fundamentally enhance the intrinsic constraint sensitivity of LLMs in planning

tasks. By leveraging the distributional divergence between model outputs under constrained and unconstrained inputs, CARL formulates an effective reward signal to improve constraint compliance without reliance on external solvers or task-specific engineering. Extensive experiments across diverse benchmarks demonstrate that CARL significantly outperforms both standard RFT baselines and state-of-the-art reasoning models. Furthermore, our analysis confirms that CARL induces a stronger focus on constraints while exhibiting robust generalization across different RL algorithms and model scales. Ultimately, this work establishes a scalable pathway for developing autonomous language agents capable of reliable, constraint-aware planning.

Limitations

First, regarding training efficiency, CARL incurs a marginal computational overhead due to the additional forward pass for unconstrained queries. However, as detailed in Appendix B.2, this cost is significantly lower than that of scaling up rollouts to achieve comparable gains. Second, our evaluation is currently confined to single-agent benchmarks. Extending CARL to complex combinatorial planning scenarios, such as multi-robot collaboration, remains a direction for future work.

Acknowledgements

This work was supported by the National Natural Science Foundation of China under Grant 42394060 and 42394064, Ant Group Research Fund, and Information Technology Center and State Key Lab of CAD&CG, Zhejiang University.

References

- Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*.
- Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, and 1 others. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.
- Soumyabrata Chaudhuri, Pranav Purkar, Ritwik Raghav, Shubhojit Mallick, Manish Gupta, Abhik Jana, and Shreya Ghosh. 2025. Tripcraft: A benchmark for spatio-temporally fine grained travel planning. *arXiv preprint arXiv:2502.20508*.
- Yongchao Chen, Jacob Arkin, Charles Dawson, Yang Zhang, Nicholas Roy, and Chuchu Fan. 2024. Autotamp: Autoregressive task and motion planning with llms as translators and checkers. In *2024 IEEE International conference on robotics and automation (ICRA)*, pages 6695–6702. IEEE.
- Zehui Chen, Weihua Du, Wenwei Zhang, Kuikun Liu, Jiangning Liu, Miao Zheng, Jingming Zhuo, Songyang Zhang, Dahua Lin, Kai Chen, and 1 others. 2023. T-eval: Evaluating the tool utilization capability step by step. *CoRR*.
- Shijian Deng, Wentian Zhao, Yu-Jhe Li, Kun Wan, Daniel Miranda, Ajinkya Kale, and Yapeng Tian. 2024. Efficient self-improvement in multimodal large language models: A model-level judge-free approach. *arXiv preprint arXiv:2411.17760*.
- Yao Fu, Dong-Ki Kim, Jaekyeom Kim, Sungryull Sohn, Lajanugen Logeswaran, Kyunghoon Bae, and Honglak Lee. 2024. Autoguide: Automated generation and selection of state-aware guidelines for large language model agents. *CoRR*.
- Jihao Gu, Yingyao Wang, Meng Cao, Pi Bu, Jun Song, Yancheng He, Shilong Li, and Bo Zheng. 2024. Token preference optimization with self-calibrated visual-anchored rewards for hallucination mitigation. *arXiv preprint arXiv:2412.14487*.
- Atharva Gundawar, Mudit Verma, Lin Guan, Karthik Valmееkam, Siddhant Bhamri, and Subbarao Kambhampati. 2024. Robust planning with llm-module framework: Case study in travel planning. *arXiv preprint arXiv:2405.20625*.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shiron Ma, Peiyi Wang, Xiao Bi, and 1 others. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.
- Yilun Hao, Yongchao Chen, Yang Zhang, and Chuchu Fan. 2024. Large language models can solve real-world planning rigorously with formal verification tools. *arXiv preprint arXiv:2404.11891*.
- Joy He-Yueya, Gabriel Poesia, Rose E Wang, and Noah D Goodman. 2023. Solving math word problems by combining language models with symbolic solvers. *arXiv preprint arXiv:2304.09102*.
- Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022a. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

- Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, and 1 others. 2022b. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.
- Xu Huang, Weiwen Liu, Xiaolong Chen, Xingmei Wang, Hao Wang, Defu Lian, Yasheng Wang, Ruiming Tang, and Enhong Chen. 2024. Understanding the planning of llm agents: A survey. *arXiv preprint arXiv:2402.02716*.
- Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. Llms can't plan, but can help planning in llm-modulo frameworks. *arXiv preprint arXiv:2402.01817*.
- Nabil A Kartam and David E Wilkins. 1990. Towards a foundation for evaluating ai planners. *AI EDAM*, 4(1):1–13.
- Timo Kaufmann, Paul Weng, Viktor Bengs, and Eyke Hüllermeier. 2024. A survey of reinforcement learning from human feedback.
- Andrew Kiruluta, Andreas Lemos, and Priscilla Burity. 2025. A self-supervised reinforcement learning approach for fine-tuning large language models using cross-attention signals. *arXiv preprint arXiv:2502.10482*.
- Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.
- Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and 1 others. 2020. Captum: A unified and generic model interpretability library for pytorch. *arxiv*. *arXiv preprint arXiv:2009.07896*, 2:5.
- Beibin Li, Konstantina Mellou, Bo Zhang, Jeevan Pathuri, and Ishai Menache. 2023. Large language models for supply chain optimization. *arXiv preprint arXiv:2307.03875*.
- Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*.
- Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward. *Advances in Neural Information Processing Systems*, 37:124198–124235.
- Allen Newell, John Calman Shaw, and Herbert A Simon. 1958. Elements of a theory of human problem solving. *Psychological review*, 65(3):151.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, and 1 others. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *arXiv preprint arXiv:2307.16789*.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741.
- Shreyas Sundara Raman, Vanya Cohen, Ifrah Idrees, Eric Rosen, Raymond Mooney, Stefanie Tellex, and David Paulius. 2024. Cape: Corrective actions from precondition errors using large language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14070–14077. IEEE.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.
- Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. 2025. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, pages 1279–1297.
- Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. 2021. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*.
- Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2022. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*.
- Karthik Valmeekam, Kaya Stechly, and Subbarao Kambhampati. 2024. Llms still can't plan; can llms? a preliminary evaluation of openai's o1 on planbench. *arXiv preprint arXiv:2409.13373*.
- Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023. Describe, explain, plan and select: Interactive planning with large language models enables open-world multi-task agents. *arXiv preprint arXiv:2302.01560*.

- Hui Wei, Zihao Zhang, Shenghua He, Tian Xia, Shijia Pan, and Fei Liu. 2025. Plangenllms: A modern survey of llm planning capabilities. *arXiv preprint arXiv:2502.11221*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.
- Yuhuai Wu, Albert Qiaoju Jiang, Wenda Li, Markus Rabe, Charles Staats, Mateja Jamnik, and Christian Szegedy. 2022. Autoformalization with large language models. *Advances in neural information processing systems*, 35:32353–32368.
- Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024a. Travelplanner: A benchmark for real-world planning with language agents. *arXiv preprint arXiv:2402.01622*.
- Jian Xie, Kexun Zhang, Jiangjie Chen, Siyu Yuan, Kai Zhang, Yikai Zhang, Lei Li, and Yanghua Xiao. 2024b. Revealing the barriers of language agents in planning. *arXiv preprint arXiv:2410.12409*.
- Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. 2023. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*.
- Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. WebShop: Towards scalable real-world web interaction with grounded language agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.
- Chao Yu, Akash Velu, Eugene Vinitisky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. 2022. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624.
- Cong Zhang, Derrick Goh Xin Deik, Dexun Li, Hao Zhang, and Yong Liu. 2024. Planning with multi-constraints via collaborative language agents. *arXiv preprint arXiv:2405.16510*.
- Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

A Prompt used for Constraint-Extraction

We herein present the prompt designed to extract implicit constraints in T-Eval, as referenced in Sec. 3.1, along with its effect on T-Eval and a sensitivity analysis of the extraction quality.

A.1 Full Prompt

We use a simple prompt to extract and isolate the constraint-relevant portions of the input query for T-Eval, as shown below:

```
You are an expert at simplifying user queries by removing specific constraints while preserving the core intent.

Your task is to remove detailed constraints and specific requirements from the user query, keeping only the main objectives and essential context.

Rules:
1. Keep the main purpose and core actions
2. Remove specific numbers, quantities, limits
3. Remove specific dates, times, or temporal constraints
4. Remove detailed specifications or precise requirements
5. Maintain the overall structure and flow of the original query
6. Keep professional context and role descriptions
7. The output should be a simplified version that captures the essence without the fine-grained constraints

Now process the following:
Original user query:
{user_prompt}

Simplified query (remove constraints but keep core intent):
```

A.2 Effect on T-Eval

We use two cases to demonstrate the effect of our prompt on T-Eval, with the extracted implicit constraints underlined, as shown below:

[Case 1]:

Constrained Queries:

As the office manager, I need to find a meeting room that is available for the next two hours for a team meeting today. Once an available room is found, please book it for the specified duration, starting from the current time.

Unconstrained Queries:

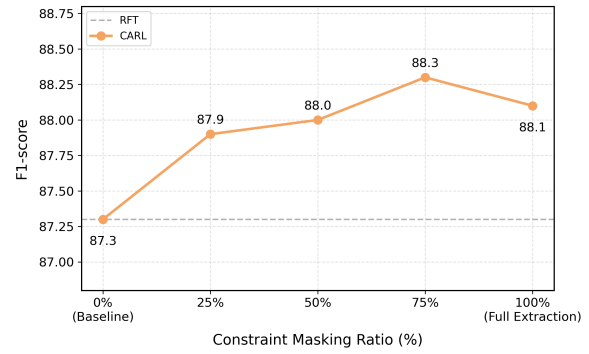


Figure 6: Sensitivity analysis of constraint extraction quality on T-Eval.

As the office manager, I need to find a meeting room for a team meeting and book it.

[Case 2]:

Constrained Queries:

I am writing a research paper on quantum computing, and I need information about the first author of the articles. Please find articles related to quantum computing and provide me with the meta information of the first three articles. Lastly, I need to know if there are any meeting rooms available tomorrow from 2:00 PM to 4:00 PM.

Unconstrained Queries:

I am writing a research paper on quantum computing, and I need information about the author of the articles. Please find articles related to quantum computing and provide me with the meta information of the articles. Lastly, I need to know if there are any meeting rooms available tomorrow.

A.3 Sensitivity Analysis of Constraint Extraction Quality

We herein investigate the sensitivity of the CARL framework to the quality of the constraint-extraction prompt. Specifically, we examine whether the performance gain relies on the exhaustive removal of constraint information from the unconstrained baseline, or if the model can acquire constraint awareness effectively from partial contrastive signals.

Settings. We introduce the *Constraint Masking Ratio* (ρ), defined as the proportion of constraint-related tokens identified and masked during the construction of the unconstrained query $x_{\setminus c}$. Using the full constraint set extracted by the prompt in Appendix A.1 as the pseudo-ground truth ($\rho = 100\%$),

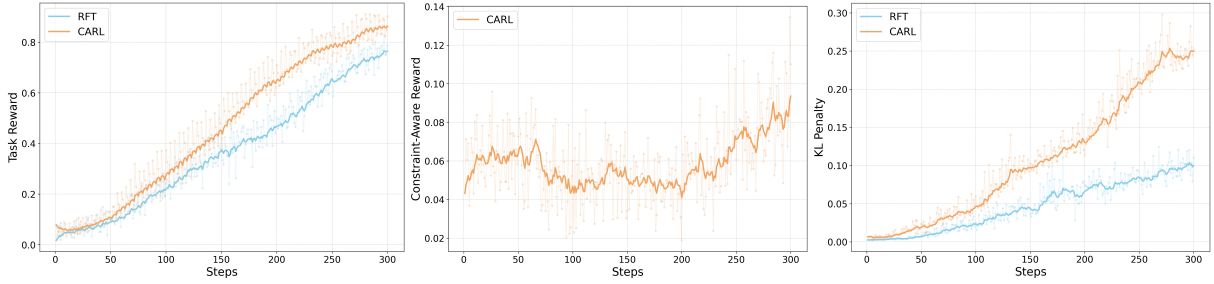


Figure 7: Comparison of the training dynamics on TravelPlanner. Solid lines indicate exponential moving averages of the data.

we simulate varying levels of extraction quality by randomly downsampling the masked constraints at ratios of $\{0\%, 25\%, 50\%, 75\%, 100\%\}$. Note that $\rho = 0\%$ corresponds to the standard RFT baseline, as it yields no contrastive signal. All experiments are conducted on the T-Eval benchmark using the Qwen3-8B model.

Results. The F1-scores on T-Eval across different masking ratios are illustrated in Figure 6. Starting from the RFT baseline ($\rho = 0\%$) with an F1-score of 87.3, applying a minimal masking ratio of $\rho = 25\%$ results in a significant performance jump to 87.9. As ρ increases, the performance exhibits a robust plateau with stochastic fluctuations: $\rho = 50\%$ yields 88.0, while $\rho = 75\%$ reaches an incidental peak of 88.3, before settling at 88.1 with full extraction ($\rho = 100\%$).

Analysis. These results demonstrate that CARL is highly robust to the completeness of constraint extraction. A mere 25% masking ratio captures the majority of the performance gain (0.6 out of 0.8), indicating that the model learns the *general mechanism* of attending to constraints via partial contrastive signals, rather than relying on precise masking every constraint token. Furthermore, the fluctuation observed between 50% and 100% suggests that once sufficient contrast is established, further optimization of the extraction prompt yields marginal or noisy returns. This confirms that CARL enables robust constraint-aware planning without strict dependency on carefully designed extraction prompts.

B Comparison with RFT

We herein compare the GRPO-version training dynamics and efficiency of CARL and RFT.

Method	Rollouts	Time per Step	Accuracy
RFT	8	406.2	73.8
	16	674.2	75.0
CARL	8	445.7	77.2

Table 5: Efficiency analysis with different rollouts on BlocksWorld. Results are reported in terms of training time per step (s) and accuracy (%).

B.1 Training Dynamics

Since we maximize a KL divergence that is theoretically unbounded, the model may “hack” the constraint-aware reward, eventually leading to performance collapse. The KL penalty, however, can mitigate the risk of reward hacking by constraining the magnitude of policy updates. As illustrated in Figure 7 on TravelPlanner, although CARL exhibits a higher KL penalty compared to RFT—reflecting more significant policy updates driven by the additional constraint supervision—the training process remains stable. The task reward of CARL consistently surpasses that of RFT, showing faster convergence and higher final performance. Meanwhile, the constraint-aware reward fluctuates within a reasonable range without divergence. This demonstrates that the KL penalty effectively balances the optimization, allowing CARL to leverage constraint-aware signals for superior planning capabilities without succumbing to reward hacking.

B.2 Training Efficiency

To evaluate training efficiency, we measure the average time per training step (in seconds) and planning accuracy across different settings (Qwen3-8B is used here). As shown in Table 5, increasing rollouts from 8 to 16 modestly improves RFT but increases about 70% more computational cost. In contrast,

Method	BlocksWorld	TravelPlanner					T-Eval		
		Commonsense		Hard		Final	Precision	Recall	F1-score
		Micro	Macro	Micro	Macro				
DeepSeek-R1-Distill-Llama-8B	1.4	61.2	0.0	0.0	0.0	0.0	81.8	79.3	79.4
+SFT	12.6	78.1	16.7	29.8	16.1	3.9	87.1	<u>86.5</u>	86.0
+RFT	<u>42.0</u>	<u>80.8</u>	<u>25.0</u>	<u>36.2</u>	<u>19.4</u>	<u>5.6</u>	<u>88.0</u>	86.0	<u>86.4</u>
+CARL (Ours)	52.6	81.1	32.1	42.9	28.9	11.7	88.4	88.6	87.5
Qwen3-8B	31.2	72.7	7.8	34.8	27.8	2.2	86.6	83.5	84.2
+SFT	14.2	77.4	25.0	24.3	14.4	8.3	84.0	84.3	83.5
+RFT	<u>73.8</u>	<u>96.3</u>	<u>74.4</u>	<u>65.7</u>	<u>48.9</u>	<u>45.0</u>	<u>88.6</u>	<u>88.2</u>	<u>87.3</u>
+CARL (Ours)	77.2	97.3	81.7	73.1	59.4	56.1	89.5	88.5	88.1

Table 6: Performance comparison with SFT on planning benchmarks. The best and second-best results are **bold** and underlined.

CARL achieves better performance with only a 10% overhead, striking a better balance between efficiency and effectiveness. This advantage is due to CARL’s design: although it generates an additional set of unconstrained queries, these require only a single forward pass for log-probabilities and are excluded from the expensive rollout process.

C Implementation Details for GRPO

We herein present more details about benchmarks, training, and inference for GRPO.

Benchmarks. For BlocksWorld, we employ the official one-shot setting. For TravelPlanner, we adopt the "sole-planning" mode to focus on the LLMs’ planning ability, excluding the influence of information gathering abilities required in the "two-stage" mode. For T-Eval, we use the default setting.

Training. For GRPO, we employ the Verl framework. Regarding hyperparameters, we set the policy LLM learning rate to $1e-6$ and sample 8 responses per query. Training is conducted on a single node with 8 A100 GPUs, with a total batch size of 64. The maximum response length is set to 8,192 tokens. To optimize GPU memory usage, we enable gradient checkpointing and use Fully Sharded Data Parallel (FSDP) with CPU offloading.

For efficient LLM rollouts, we adopt vllm with a tensor parallel size of 1 and GPU memory utilization ratio of 0.6. The rollout sampling uses a temperature of 0.6 and a top-p value of 1.0. The weighting coefficients α and β (see Equation 7) are both set to 0.001.

We train the model for 150 steps on BlocksWorld and T-Eval, and 300 steps on the more challenging

TravelPlanner. The task reward is computed using the evaluation script provided by each benchmark.

Inference. During inference, we employ vllm with a sampling temperature of 0.6 and set the maximum response length uniformly to 30,000 tokens to prevent truncation.

D Additional Experimental Results

We herein present additional experimental results, including comparisons with SFT, prompting, and agent-based methods, generalization performance on other benchmarks, and more results and additional analyses on BlocksWorld.

D.1 Comparison with SFT

Table 6 shows the performance comparison with SFT on planning benchmarks.

Analysis. The experimental results demonstrate that RL-based methods are more effective than SFT at improving planning capabilities. We attribute this to two primary reasons. First, the ground-truth data provided by these benchmarks often contains only the final answer without the intermediate reasoning process. Consequently, direct SFT is not conducive to the model’s ability to "think" and learn high-quality planning patterns. Second, planning tasks may have multiple feasible solutions that satisfy the given constraints, yet the ground truth typically provides only one. SFT may thus cause the model to memorize this specific ground-truth solution rather than genuinely learning how to plan.

D.2 Comparison with Prompting-based Methods

We herein introduce three prompting-based methods for planning performance comparison. Fol-

Method	BlocksWorld	TravelPlanner
<i>Prompting (Base: GPT-4o)</i>		
Direct	42.4	7.8
Behavioral Cloning	51.2	9.0
Oracle Feedback	52.0	11.7
Reference	51.4	8.3
<i>RL Training (Base: Qwen3-8B)</i>		
CARL (Ours)	77.2	56.1

Table 7: Performance comparison with prompting-based methods on planning benchmarks. Metrics: accuracy for BlocksWorld and final pass rate for TravelPlanner.

lowing previous work (Zhao et al., 2024; Fu et al., 2024), these methods summarize insights from previous attempts to guide the agent:

- Behavioral Cloning: The agent is provided with previous failed attempts along with a ground truth plan.
- Oracle Feedback: The agent is provided with previous failed attempts accompanied by feedback from a solver or evaluator that explicitly explains the reasons for failure.
- Reference: This setting utilizes human-written insights to serve as a ground truth summary of constraints.

Analysis. We compare these prompting strategies, implemented using the powerful GPT-4o, against our CARL framework trained on the much smaller Qwen3-8B. As presented in Table 7, while prompting methods like oracle feedback improve over the direct baseline, they remain significantly inferior to CARL. Notably, CARL achieves a 77.2% accuracy on BlocksWorld and a 56.1% final pass rate on TravelPlanner, far surpassing even the best prompting results (52.0% and 11.7%, respectively). This demonstrates that while in-context guidance provides marginal gains, CARL effectively internalizes constraint awareness, allowing a smaller model to outperform larger models relying on extensive prompt engineering.

D.3 Comparison with Agent-based Methods

We herein introduce three agent-based methods for planning performance comparison:

- LLM-Modulo (ICML 2024): A planning agent based on feedback from external constraint critics from Kambhampati et al. (2024).

- Multi-Agent (COLING 2025): A collaborative multi-agent system for planning based on task decomposition from Zhang et al. (2024).
- SMT Solver (NAACL 2025): A planning agent based on results of an external optimization solver from Hao et al. (2024).

Results. We chose the complex real-world planning task TravelPlanner for comparison. Notably, TravelPlanner operates in two modes: "sole-planning" and "two-stage". The former focuses on LLMs' planning ability, with all relevant travel information provided as a lengthy textual context within the prompt, while the latter requires LLMs to gather information and complete planning. The aforementioned agent-based methods are specifically designed with tailored prompts for the "two-stage" mode and cannot be directly adapted to the "sole-planning" mode (which we use in the main results). We therefore report their performance under the "two-stage" mode, with results shown in Table 8.

Analysis. The experimental results demonstrate that while current agent-based methods (e.g., LLM-Modulo and Multi-Agent system) exhibit strong performance in complex planning tasks, their effectiveness typically relies on top models like GPT-4 combined with external tools (e.g., constraint critics) or multi-agent collaboration mechanisms to complete reasoning and planning processes. In contrast, our proposed CARL framework differs fundamentally in design philosophy: it neither depends on top models nor introduces external solvers or multi-agent architectures, but instead achieves end-to-end scalable constraint-aware planning within a single lightweight model.

Notably, using only the Qwen3-8B model – a medium-scale model – CARL still outperforms two of the three agent-based baselines (LLM-Modulo and Multi-Agent). This indicates that by internalizing constraint comprehension within the model's planning process, CARL effectively compensates for its scale limitations while demonstrating robust planning capabilities.

Although the SMT solver-based method currently achieves state-of-the-art performance, its superiority heavily depends on a tightly-coupled GPT-4 + external solver architecture. We observe significant performance degradation when replacing GPT-4 with the equally capable 123B Mistral-Large model, revealing excessive dependency on

Method	Base Model	External Solver	TravelPlanner				Final
			Commonsense		Hard		
			Micro	Macro	Micro	Macro	
LLM-Modulo Multi-Agent	GPT-4	No	-	-	-	-	20.0
			90.0	41.7	55.7	48.3	31.7
SMT Solver	GPT-4	Yes	95.0	95.0	95.7	98.9	93.3
	Mistral-Large		72.0	70.6	63.3	66.7	66.7
CARL (Ours)	Qwen3-8B	No	97.3	81.7	73.1	59.4	56.1

Table 8: Performance comparison with agent-based methods on TravelPlanner.

specific architectures (particularly GPT-4). **Furthermore, this method requires external solvers to search through vast solution spaces, incurring substantial latency (averaging 245 seconds per query on TravelPlanner) – over 24× slower than CARL’s inference efficiency (10s average).** Such high latency and strong dependencies limit its practical scalability and deployment flexibility.

Given the fundamental differences between CARL and existing agent-based methods in system architecture, model dependency, planning workflow, and external tool usage, direct "end-to-end" performance comparisons may lack fairness and interpretability due to inconsistent experimental setups. We therefore exclude direct performance comparisons with these methods in the main results to ensure experimental rigor and comparability.

Nevertheless, it must be emphasized that CARL achieves planning capabilities comparable to or even surpassing those of GPT-4-based agent methods with complex engineering architectures – all without external solvers, top models, or exceeding 8B parameters. This breakthrough highlights CARL’s significant potential in empowering medium- and small-scale language models to achieve autonomous, efficient, and compliant planning, while establishing a new technical pathway for developing independent, lightweight, and reliable intelligent agents.

D.4 Generalization Performance on Other Benchmarks

We herein introduce three additional benchmarks to verify the generalization ability of CARL. ALFWorld and WebShop serve as **multi-turn interactive tasks without explicit constraints** to test generalization from BlocksWorld, while TripCraft serves as **a more complex travel planning task** to

test generalization from TravelPlanner.

- ALFWorld (ICLR 2021): An embodied environment designed to assess the ability of LLM agents to perform multi-step decision-making (Shridhar et al., 2021). In each episode, the agent receives a text goal and must accomplish it through multi-turn interaction with the environment. It includes 3,827 task instances across six categories of common household activities.
- WebShop (NeurIPS 2022): A complex, web-based interactive environment designed to test LLM agents in realistic online shopping scenarios (Yao et al., 2022). To complete the task, the agent must interact with a simulated HTML-based shopping website to search for, navigate to, and ultimately purchase a suitable item from over 1.1 million products.
- TripCraft (ACL 2025): A real-world travel planning benchmark constructed entirely from real data sources to ensure geographical coherence (Chaudhuri et al., 2025). Unlike previous datasets reliant on semi-synthetic data, TripCraft integrates complex real-world constraints such as public transit schedules and event availability. This makes it a more challenging and realistic testbed for evaluating the generalization capabilities of planning agents.

Settings. We evaluate the cross-domain generalization of models trained on specific tasks. First, we take the models trained on BlocksWorld and evaluate them on ALFWorld and WebShop without any fine-tuning. For ALFWorld, we report the success rate (%) for each of the six subtasks as well as the overall result. For WebShop, we report both the score and the success rate (%). Second, to test

Method	ALFWorld							WebShop	
	Pick	Look	Clean	Heat	Cool	Pick2	All	Score	Succ.
Qwen3-8B	58.6	33.3	12.9	10.5	4.3	10.0	21.9	7.1	0.8
+RFT	55.2	50.0	6.5	10.5	0.0	5.0	18.8	5.3	0.8
+CARL	58.6	66.7	9.7	0.0	4.3	10.0	21.1	8.1	1.6

Table 9: Generalization performance from BlocksWorld to ALFWorld and WebShop.

Method	TripCraft				
	Commonsense		Hard		Final
	Micro	Macro	Micro	Macro	
Qwen3-8B	90.8	0.0	17.3	14.8	0.0
+RFT	90.1	1.8	23.2	21.3	0.2
+CARL	94.8	2.9	27.3	25.7	1.1

Table 10: Generalization performance from TravelPlanner to TripCraft.

generalization in planning scenarios, we evaluate the models trained on TravelPlanner on the more realistic TripCraft benchmark. We adopt the "w/o Parameter Info" and "3-day" settings for TripCraft and report the micro/macro scores for commonsense and hard constraints, along with the final pass rate.

Analysis. Results in Table 9 and Table 10 reveal two key insights regarding generalization. First, enhanced planning capabilities contribute to robust multi-step decision-making. Standard RFT tends to overfit the source domain (BlocksWorld), leading to negative transfer on unseen interactive tasks (e.g., performance drops on ALFWorld compared to the base model). In contrast, CARL not only mitigates this degradation on ALFWorld but surprisingly outperforms both RFT and the base model on WebShop. This suggests that the constraint-aware optimization in CARL does not merely memorize specific constraints; instead, it refines the model’s fundamental reasoning capability. By learning to adhere to constraints, the model becomes more deliberate in its planning, which translates effectively into better performance in complex, multi-turn decision-making environments.

Second, CARL demonstrates superior transferability in complex planning scenarios. On TripCraft, which shares the planning nature of TravelPlanner but imposes significantly harder real-world constraints, CARL achieves a substantial improvement over baselines (e.g., a 25.7 macro

Method	Qwen-7B	Qwen-14B
RFT	24.6	86.0
CARL	29.8	93.0

Table 11: More results with DeepSeek-R1-Distill models on BlocksWorld.

score on hard constraints vs. 14.8 for base). This confirms that CARL successfully internalizes a generalized constraint-aware mechanism that remains effective across distinct planning domains.

D.5 More Results on BlocksWorld

We apply CARL to other DeepSeek-R1-Distill models and compare it against RFT. As shown in Table 11, CARL consistently outperforms RFT. These results, combined with the performance of DeepSeek-R1-Distill-Llama-8B in Table 1, suggest that CARL can generalize to different model architectures. Notably, CARL achieves performance on par with top models at the 14B scale on BlocksWorld.

D.6 Additional Analyses on BlocksWorld

To further validate the practical behavior of CARL, we provide two additional compact analyses: threshold filtering and robustness to noisy task rewards.

Threshold Filtering Ablation. We test whether filtering low-divergence samples improves performance. For CA-DPO (1,600 pairs), we filter out the bottom- $k\%$ pairs by KL divergence during pair construction. For CA-GRPO, we apply a hard mask to R_{CA} for the bottom- $k\%$ samples in each batch. Results are shown in Table 12. Small thresholds do not provide consistent gains, while aggressive filtering degrades performance, indicating that CARL already self-regulates effectively without hard filtering.

Noise Injection Robustness. To simulate imperfect reward signals, we randomly flip the binary

Threshold	CA-DPO	CA-GRPO
0%	53.8	77.2
0.5%	53.4	77.2
1.0%	53.6	76.8
2.0%	53.0	77.0
5.0%	52.6	76.6

Table 12: Effect of threshold filtering on BlocksWorld.

Noise Ratio	RFT	CARL	Difference
0%	73.8	77.2	+3.4
5%	69.6	76.2	+6.6
10%	66.4	74.4	+8.0
15%	59.8	67.6	+7.8
20%	50.2	56.4	+6.2

Table 13: Noise injection robustness on BlocksWorld.

task reward during GRPO training on BlocksWorld (Qwen3-8B). Results are shown in Table 13. CARL maintains consistent advantages under all tested noise levels, suggesting stronger robustness when external reward signals are imperfect.

E Implementation Details for DPO

We herein provide more details about preference pair construction and training for DPO.

Preference Pair Construction. For Naive-DPO, we input constrained queries and construct preference pairs based on the correctness of the responses. For CA-DPO, we input both constrained and unconstrained queries. The correct responses generated for constrained queries are selected as positive samples, while the responses generated for unconstrained queries are randomly chosen as negative samples, which are then utilized to construct preference pairs. Our sampling strategies for preference pair construction can be summarized as shown in Table 14.

Training. For DPO, we employ the LLaMA-Factory framework. We consistently train the model for 3 epochs with a batch size of 64, a learning rate of $2.0e-6$, a preference coefficient β of 0.1, and a warmup ratio of 0.05 on BlocksWorld.

F Sensitivity Analysis of Constraint-Aware Reward Weight

We herein investigate the sensitivity of the CARL framework to the weighting coefficient α of the

Sampling Strategy	Chosen Samples	Rejected Samples
Naive-DPO	$y^{(i)} \in \mathcal{Y}_{\text{valid}}(x^{(i)})$	$y^{(i)} \notin \mathcal{Y}_{\text{valid}}(x^{(i)})$
CA-DPO	$y^{(i)} \in \mathcal{Y}_{\text{valid}}(x^{(i)})$	$y_{\setminus C}^{(i)}$

Table 14: The sampling strategies used in Naive-DPO and CA-DPO. In CA-DPO, the rejected samples are outputs generated under conditions where constraints are absent, represented as $y_{\setminus C}^{(i)}$. This design choice aims to maximize the divergence between $y_{\setminus C}^{(i)}$ and the valid constrained output $y^{(i)}$. By doing so, the model is better guided to understand and incorporate the crucial role played by constraints in its learning process.

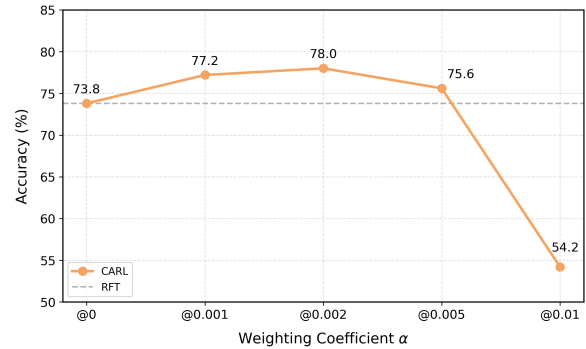


Figure 8: Sensitivity analysis of constraint-aware reward weight α on BlocksWorld. The KL penalty coefficient is fixed at $\beta = 0.001$.

constraint-aware reward.

Settings. We conduct the ablation study on the BlocksWorld benchmark using the Qwen3-8B model. While keeping the KL penalty coefficient fixed at $\beta = 0.001$, we vary α across the set $\{0, 0.001, 0.002, 0.005, 0.01\}$. Note that $\alpha = 0$ is equivalent to the standard RFT baseline. We report the accuracy for each setting to analyze the impact of α on model performance.

Analysis. The results, illustrated in Figure 8, demonstrate that CARL is highly robust to the selection of α . The method consistently outperforms the RFT baseline across a broad range of values, from $\alpha = 0.001$ to $\alpha = 0.005$. Notably, while our main experiments adopted a conservative setting of $\alpha = 0.001$, the performance peaks at $\alpha = 0.002$, and even at $\alpha = 0.005$ —a five-fold increase—the model maintains a competitive accuracy of 75.6%. This indicates that CARL does not require precise hyperparameter tuning to yield improvements. However, performance degrades far below the baseline at $\alpha = 0.01$, suggesting that an excessively strong constraint signal overwhelms

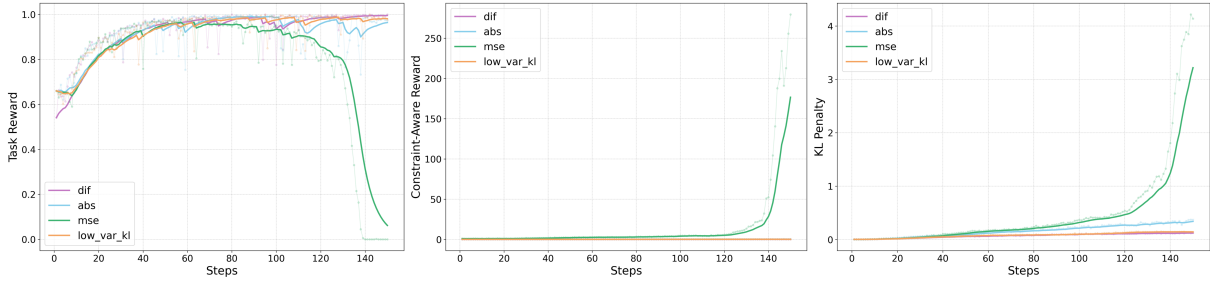


Figure 9: Comparison of the training dynamics on BlocksWorld. Solid lines indicate exponential moving averages of the data.

the primary task objectives.

G Impact of Reward Calculation Strategy on Training Dynamics

The results from the ablation study, presented in Table 4, clearly show that the choice of reward calculation metric has a profound impact on training outcomes. While the dif, abs, and low_var_kl strategies all yield high and comparable final performance (76.6, 76.0, and 77.2, respectively), the mse strategy leads to a complete training collapse, with the model achieving a near-zero score of 0.2.

The training dynamics, illustrated in Figure 9, reveal the underlying reasons for this disparity. The mse strategy (green line) initially shows a promising increase in task reward, but it abruptly collapses after approximately 120 steps. This collapse is directly correlated with an explosive, uncontrolled spike in both the constraint-aware reward and the KL penalty, indicating severe training instability. We hypothesize that the quadratic nature of mse creates a volatile reward signal, excessively penalizing larger deviations and leading to policy divergence.

In contrast, dif, abs, and low_var_kl all maintain a stable and high task reward throughout training. Their constraint-aware rewards remain close to zero, providing a consistent and stable learning signal. Notably, low_var_kl and dif also maintain the lowest KL penalty, suggesting they guide the policy towards improvement more efficiently without deviating drastically from the reference model. This combination of high final performance and superior training stability confirms that low_var_kl is the most robust and effective reward calculation strategy for our framework.

H Details for Attribution Analysis

Setup. In this paper, we adopt Feature Ablation (implemented via FeatureAblation in Cap-

tum (Kokhlikyan et al., 2020)) as our analysis strategy for testing the inner workings of LLMs when planning. For TravelPlanner, we conduct analysis using the entire validation set (180 samples). For BlocksWorld, we randomly select 200 samples from the test set due to the high computational cost.

Case Study. We present several cases used in our attribution analysis. The model after RFT fails in planning due to its insufficient sensitivity to constraints, whereas the model after CARL succeeds in planning by adequately attending to the constraints. Due to the length of the full response, we present only the portion following the "`</think>`" identifier.

The case on BlocksWorld demonstrates that the model after RFT executes a pick-up action immediately following an unstack operation, violating the fundamental constraint that prohibits pick-up or unstack operations while holding a block. The related constraints are explicitly highlighted with underlines within the case.

The case on TravelPlanner demonstrates that the model after CARL proactively considers whether daily accommodation and meal arrangements align with user preferences, thus producing plans that fully satisfy all constraints. This focus on preferences is partially explicitly highlighted with double underscores within the case.

[BlocksWorld]

Query:

I am playing with a set of blocks where I need to arrange the blocks into stacks. Here are the actions I can do

Pick up a block
Unstack a block from on top of another block
Put down a block
Stack a block on top of another block

I have the following restrictions on my actions:
I can only pick up or unstack one block at a time.
I can only pick up or unstack a block if my hand is empty.

I can only pick up a block if the block is on the table and the block is clear. A block is clear if the block has no other blocks on top of it and if the block is not picked up.

I can only unstack a block from on top of another block if the block I am unstacking was really on top of the other block.

I can only unstack a block from on top of another block if the block I am unstacking is clear.

Once I pick up or unstack a block, I am holding the block.

I can only put down a block that I am holding.

I can only stack a block on top of another block if I am holding the block being stacked.

I can only stack a block on top of another block if the block onto which I am stacking the block is clear.

Once I put down or stack a block, my hand becomes empty.

Once you stack a block on top of a second block, the second block is no longer clear

.

[STATEMENT]

As initial conditions I have that, the blue block is clear, the hand is empty, the red block is on top of the orange block, the blue block is on top of the yellow block, the yellow block is on top of the red block and the orange block is on the table.

My goal is to have that the blue block is on top of the red block and the yellow block is on top of the orange block.

My plan is as follows:

[PLAN]

unstack the blue block from on top of the yellow block
put down the blue block
unstack the yellow block from on top of the red block
stack the yellow block on top of the blue block
unstack the red block from on top of the orange block
put down the red block
unstack the yellow block from on top of the blue block
stack the yellow block on top of the orange block
pick up the blue block
stack the blue block on top of the red block
[PLAN END]

[STATEMENT]

As initial conditions I have that, the red block is clear, the orange block is clear, the hand is empty, the red block is on top of the yellow block, the orange block is on top of the blue block, the blue block is on the table and the yellow block is on the table.

My goal is to have that the blue block is on top of the yellow block and the yellow block is on top of the red block.

My plan is as follows:

[PLAN]

RFT Response:

The user's plan contains errors due to misalignment with the initial conditions. Here's the corrected plan:

Correct Plan:

1. Unstack the red block from the yellow block
 - Red is on top of yellow; red is clear (no blocks above it).
 - Unstack red and place it on the table.
2. Pick up the yellow block
 - Yellow is now on the table (after red is removed) and is clear.
3. Stack the yellow block on the red block
 - Red is on the table (clear), so yellow can be stacked on it.
 - Result: Red → Yellow.
4. Unstack the orange block from the blue block
 - Orange is on top of blue; orange is clear (no blocks above it).
 - Unstack orange and place it on the table.
5. Pick up the blue block
 - Blue is now on the table (after orange is removed) and is clear.
6. Stack the blue block on the yellow block
 - Yellow is on top of red and is clear (no blocks above it).
 - Result: Red → Yellow → Blue.

Final Arrangement:

- Red block is on the table.
- Yellow block is on top of red.
- Blue block is on top of yellow.

This sequence adheres to all constraints and achieves the goal. The user's original plan incorrectly assumed blue was on yellow initially, which was not the case.

CARL Response:

To achieve the goal of having blue on top of yellow and yellow on top of red, given the initial configuration:

- Red is on top of yellow
- Orange is on top of blue
- Blue and yellow are on the table
- Red and orange are clear

We need to rearrange the blocks while adhering to the rules of block manipulation. Here's the step-by-step plan:

Plan

1. Unstack the red block from on top of the yellow block
 - Red is clear (initial condition), and it is on top of yellow.
 - After unstacking, red is held in the hand.
 - Yellow is now clear (since red is removed from above it).
2. Put down the red block
 - Hand becomes empty.
 - Red is now on the table.
3. Pick up the yellow block
 - Yellow is now clear (no blocks on top).
 - Hand is empty, so this is allowed.
4. Stack the yellow block on top of the red block
 - You are holding yellow, and red is clear (before stacking).
 - After stacking, red is no longer clear (rule: stacking makes the base block non-clear).
 - Hand becomes empty.
5. Unstack the orange block from on top of the blue block
 - Orange is clear (initial condition), and it is on top of blue.

- After unstacking, orange is held in the hand.
 - Blue is now clear (since orange is removed from above it).
6. Put down the orange block
- Hand becomes empty.
 - Orange is now on the table.
7. Pick up the blue block
- Blue is now clear (no blocks on top).
 - Hand is empty, so this is allowed.
8. Stack the blue block on top of the yellow block
- You are holding blue, and yellow is clear (no blocks on top).
 - After stacking, yellow is no longer clear (rule: stacking makes the base block non-clear).
 - Hand becomes empty.

Final Arrangement

- Red block is on the table.
- Yellow block is on top of red.
- Blue block is on top of yellow.

This satisfies the goal: blue is on top of yellow, and yellow is on top of red.

Answer

[PLAN]

```
unstack the red block from on top of the yellow block
put down the red block
pick up the yellow block
stack the yellow block on top of the red block
unstack the orange block from on top of the blue block
put down the orange block
pick up the blue block
stack the blue block on top of the yellow block
```

[PLAN END]

[TravelPlanner]

Query:

I'm looking for a 7-day travel itinerary for 2 people, starting from Reno and heading to Texas, specifically visiting 3 different cities. The travel dates are from March 7th to March 13th, 2022, with a set budget of \$4,300. We require accommodations that adhere to house rules regarding visitors and should ideally be entire rooms. For food, we would love to try a variety of cuisines, including Chinese, French, American, and Mediterranean.

RFT Response:

Travel Plan:

Day 1:

Current City: from Reno to Abilene

Transportation: Self-driving, from Reno to Abilene, Duration: 22 hours 27 mins, Distance: 2,412 km, Cost: 120

Breakfast: Thai Garden, Abilene

Attraction: The Grace Museum, Abilene

Lunch: Crispy Crust, Abilene

Dinner: Mx Corn, Abilene

Accommodation: Apartment minutes from manhattan, Abilene

Day 2:

Current City: Abilene

Transportation: -

Breakfast: Cakes Degree, Abilene

Attraction: Frontier Texas!, Abilene; Historic Fort Phantom Hill, Abilene

Lunch: LPK Waterfront, Abilene

Dinner: Pawan Foods, Abilene

Accommodation: Apartment minutes from manhattan, Abilene

Day 3:

Current City: from Abilene to Amarillo

Transportation: Self-driving, from Abilene to Amarillo, Duration: 4 hours 10 mins, Distance: 446 km, Cost: 22

Breakfast: Wood Box Cafe, Amarillo

Attraction: Cadillac Ranch, Amarillo; Amarillo Botanical Gardens, Amarillo

Lunch: Anand Restaurant, Amarillo

Dinner: Thalaivar, Amarillo

Accommodation: Trendy Brooklyn Room - 20mins from Manhattan, Amarillo

Day 4:

Current City: Amarillo

Transportation: -

Breakfast: The Cinnamon Kitchen, Amarillo

Attraction: Amarillo Zoo, Amarillo; Don Harrington Discovery Center, Amarillo

Lunch: Punjabi Chaap Corner, Amarillo

Dinner: Cafe Coffee Day, Amarillo

Accommodation: Trendy Brooklyn Room - 20mins from Manhattan, Amarillo

Day 5:

Current City: from Amarillo to Lubbock

Transportation: Self-driving, from Amarillo to Lubbock, Duration: 1 hour 47 mins, Distance: 197 km, Cost: 9

Breakfast: Grand Barbeque Buffet Restaurant, Lubbock

Attraction: Buddy Holly Center, Lubbock; National Ranching Heritage Center, Lubbock

Lunch: Kapoor's Sanjha Chulha, Lubbock

Dinner: San Carlo, Lubbock

Accommodation: Gorgeous Spacious Room in Clinton Hill, Lubbock

Day 6:

Current City: Lubbock

Transportation: -

Breakfast: Assam Tea Corner, Lubbock

Attraction: American Windmill Museum, Lubbock; Lubbock Lake National Historic Landmark, Lubbock

Lunch: Domino's Pizza, Lubbock

Dinner: Platinum Lounge, Lubbock

Accommodation: Gorgeous Spacious Room in Clinton Hill, Lubbock

Day 7:
Current City: from Lubbock to Reno
Transportation: Self-driving, from Lubbock to Reno, Duration: 20 hours 3 mins,
Distance: 2,145 km, Cost: 107
Breakfast: -
Attraction: -
Lunch: -
Dinner: -
Accommodation: -

CARL Response:

Travel Plan:

Day 1:

Current City: from Reno to Abilene
Transportation: Self-driving, from Reno to Abilene, duration: 22 hours 27 mins,
distance: 2,412 km, cost: 120
Breakfast: -
Attraction: -
Lunch: -
Dinner: Thai Garden, Abilene (Average Cost: \$13, Cuisines: Cafe, American, Desserts)

Accommodation: NYC Studio near Central Park and the Hudson River, Abilene (price: \$304, room type: Entire home/apt, house rules: No pets, minimum nights: 2)

Day 2:

Current City: Abilene
Transportation: -
Breakfast: Crispy Crust, Abilene (Average Cost: \$54, Cuisines: Desserts, Bakery, Cafe, American, Seafood)
Attraction: The Grace Museum, Abilene
Lunch: Mx Corn, Abilene (Average Cost: \$62, Cuisines: Tea, Cafe, Italian)

Dinner: The Grand Trunk Road, Abilene (Average Cost: \$80, Cuisines: Tea, Pizza, Bakery, BBQ, Chinese, Mediterranean)
Accommodation: NYC Studio near Central Park and the Hudson River, Abilene

Day 3:

Current City: from Abilene to Amarillo
Transportation: Self-driving, from Abilene to Amarillo, duration: 4 hours 10 mins,
distance: 446 km, cost: 22
Breakfast: -
Attraction: -
Lunch: -
Dinner: Wood Box Cafe, Amarillo (Average Cost: \$11, Cuisines: Desserts, French, Fast Food, Cafe, American)
Accommodation: Spacious retreat, Amarillo (price: \$528, room type: Entire home/apt, house rules: No smoking, minimum nights: 2)

Day 4:

Current City: Amarillo
Transportation: -
Breakfast: Komachi, Amarillo (Average Cost: \$71, Cuisines: Desserts, Tea, Fast Food, Chinese, Mediterranean)
Attraction: Cadillac Ranch, Amarillo
Lunch: Anand Restaurant, Amarillo (Average Cost: \$53, Cuisines: Fast Food, Seafood, Desserts, Italian)
Dinner: Shahi Food Corner, Amarillo (Average Cost: \$90, Cuisines: French, Pizza, Bakery, Mediterranean)
Accommodation: Spacious retreat, Amarillo

Day 5:

Current City: from Amarillo to Lubbock
Transportation: Self-driving, from Amarillo to Lubbock, duration: 1 hour 47 mins,
distance: 197 km, cost: 9
Breakfast: -
Attraction: -
Lunch: -
Dinner: Annapurna Bhandar, Lubbock (Average Cost: \$78, Cuisines: French, Chinese, Desserts, Fast Food)
Accommodation: Cozy Clean Small Apartment, Lubbock (price: \$313, room type: Entire

home/apt, house rules: No children under 10 & No parties, minimum nights: 2)

Day 6:

Current City: Lubbock

Transportation: -

Breakfast: Paris 6 Classique, Lubbock (Average Cost: \$91, Cuisines: Desserts, Cafe, Pizza, Mexican, BBQ, Chinese, Seafood)

Attraction: Buddy Holly Center, Lubbock

Lunch: Kapoor's Sanjha Chulha, Lubbock (Average Cost: \$21, Cuisines: Tea, Cafe, Indian, American)

Dinner: Spezia Deliveries, Lubbock (Average Cost: \$64, Cuisines: Desserts, Italian, Bakery, Cafe, Mediterranean, Seafood)

Accommodation: Cozy Clean Small Apartment, Lubbock

Day 7:

Current City: from Lubbock to Reno

Transportation: Self-driving, from Lubbock to Reno, duration: 20 hours 3 mins, distance: 2,145 km, cost: 107

Breakfast: -

Attraction: -

Lunch: -

Dinner: -

Accommodation: -