MAPS: Margin-Aware Priors and Verifier-Guided Search for Embodied Planning

Xin Li
xli494@connect.hkust-gz.edu.cn
HKUST (GZ), Guangzhou, China

Junquan Huang
jhuang635@connect.hkust-gz.edu.cn
HKUST (GZ), Guangzhou, China

Xujia Li *
leexujia@ust.hk
HKUST, Hong Kong SAR, China

Lei Chen
leichen@cse.ust.hk
HKUST (GZ) & HKUST, Guangzhou, China

Abstract

Large language models (LLMs) have shown strong reasoning capabilities in embodied planning tasks. However, they often fail to achieve good results on long-horizon and multi-robot tasks under strict physical constraints, and generate smooth but infeasible trajectories. In real-world robotic hardware deployment, systems also face two fundamental challenges: First, the fine-tuning method for open-source models, Direct Preference Optimization (DPO), typically only makes a binary true or false judgment on the planning results, without distinguishing the severity of the planning error. As a result, they fail to impose effective penalties for catastrophic violations of physical constraints. Second, methods that search over multiple candidate plans during inference do not filter out invalid plans in advance, often leading to combinatorial explosion and high inference latency, which makes them difficult to apply to real-time robotic operations. To address these limitations, we propose a complete training-to-inference architecture consisting of two modules. First, we introduce Margin-Aware Direct Preference Optimization (mDPO) during training, which distinguishes the severity of errors in negative samples, enabling the fine-tuned model to avoid generating severe errors that would render a task irreparable. Second, we deploy Verifier-Guided Search (VGS) during inference, using a symbolic verifier to hard-prune invalid search paths, thereby repairing infeasible task plans and reducing inference latency. In addition, latency analysis shows that our approach reduces the average end-to-end runtime from 12.814s to 10.969s compared with retry-based repair, while maintaining higher final success, making it more suitable for time-sensitive robotic deployment. Our code is open-sourced at https://github.com/AmberGTP5/MAPS.

1 Introduction

Embodied artificial intelligence (AI) studies how agents perceive, reason, and act in physical environments [Huang et al., 2022, ichter et al., 2023]. In recent years, large language models (LLMs) have gradually become powerful planners for embodied tasks, using their rich world knowledge to decompose high-level instructions into executable subgoals [Xi et al., 2025, Wang et al., 2023]. From a formal perspective, embodied planning is often viewed as a partially observable Markov decision process (POMDP), which requires agents to generate effective actions under strict physical and task constraints. However, the partial observability, long-horizon dependencies, and strict physical constraints in POMDP settings still make LLM-based embodied planning highly challenging in real

*corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

tasks, especially for open-source models with limited capacity. Although some progress has been made in enabling embodied planning with large open-source models, deploying LLM-based planners in multi-step, long-horizon tasks still faces critical challenges. Their auto-regressive property often leads to execution hallucinations, in which the generated plans are linguistically fluent but physically infeasible to execute [Valmeekam et al., 2023, Xie et al., 2025].

Current mainstream solutions fall into two categories: **the fine-tuning-based methods with Direct Preference Optimization** (DPO) [Yang et al., 2024, Rafailov et al., 2023] and **inference-time search methods that search over candidate planning branches or action sequences during decoding** [Yao et al., 2023a, Hao et al., 2023]. However, both of these mainstream solutions suffer from significant limitations in real-world physical environments and therefore cannot fundamentally resolve **the conflict between execution hallucinations and practical implementation constraints**:

First, DPO-based fine-tunes models use only positive and negative plan pairs, which means it treats planning failures of different severities as the same negative outcome. For example, DPO does not distinguish between the errors that only affect output formatting and the errors that make a plan physically infeasible. In a delivery task, the model may generate a plan with the correct pickup-navigation-delivery intent but express it in malformed JSON or omit the required '<answer>' block. Such errors are formatting errors that only affect parseability and do not alter the underlying execution logic. By contrast, the model may send a 'delivery drone' into an indoor room, or require a robot to carry an item beyond its payload limit. These errors violate physical constraints and can directly lead to irreversible task failure. When both kinds of errors are treated as equally negative, DPO applies the same optimization pressure to superficial formatting mistakes and severe physical violations, without prioritizing the avoidance of more serious errors. This severity-agnostic alignment approach cannot produce robust and physically consistent policies, even after fine-tuning, the model still cannot truly understand hard physical constraints, and the **task success rate remains low**.

Second, inference-time search methods essentially use tree search to systematically explore and evaluate all possible action branches at each planning node, and then select the best output based on the search results. However, these methods lack task rules and physical constraints [Stechly et al., 2024]. Therefore, they cannot filter out obviously invalid action branches before the search begins, and thus have to traverse an exponentially growing state space, which **greatly increases both computational cost and inference time**. This will lead to a sharp increase in per-step inference time and a surge in GPU memory usage, making it difficult to satisfy the millisecond-level real-time requirements of dynamic multi-robot collaboration [Snell et al., 2025].

To address these limitations, we propose **MAPS, a unified framework with Margin-Aware Priors and Verifier-Guided Search (VGS) for Embodied Planning**. Specifically, our framework consists of two stages: In the training stage, we introduce Margin-aware Direct Preference Optimization (mDPO). ODPO sets an offset from the reward difference between two responses, while mDPO uses the verifier's assessment of plan error severity to set a margin within a fixed range and give more weight to examples with more serious errors [Amini et al., 2024]. Instead of treating all failed plans as equally negative, mDPO uses verifier feedback to assess the severity of planning errors and converts this severity into different optimization margins for different negative plans. Plans that violate hard physical constraints receive larger margins than plans with only minor or repairable errors. This design encourages the model to focus more strongly on avoiding irreversible failures. In the inference stage, we further introduce VGS as a closed-loop error-correction mechanism. Given an initial plan produced by the LLM planner, a symbolic verifier checks state transitions step by step and removes infeasible branches before they are expanded. Without this pruning, the number of candidate branches grows rapidly as the search depth increases. By removing invalid branches early, VGS keeps the search focused on a much smaller set of feasible actions. When the initial plan is invalid, VGS explores only these feasible next steps and repairs the plan step by step. In this way, our framework combines better training with correction during inference, improving both reliability and efficiency in embodied planning.

To reliably reflect the main difficulties of planning under multi-step, long-horizon, and multi-robot collaboration settings, we further release **App-200, an embodied planning benchmark** built on a four-stage data construction pipeline, consisting of candidate plan generation, symbolic verification, preference filtering, and paired training sample construction. This benchmark contains 200 execution-verified tasks, and each task requires multiple embodied agents about 12 steps on average to complete. Unlike existing datasets, which mainly provide static trajectory matching and final success labels,

App-200 further includes step level error severity labels. These labels distinguish trivial formatting errors, repairable planning errors, and irrecoverable physical constraint violations. This design enables the dataset not only to evaluate task success, but also to analyze the causes of failures. Besides App-200, we evaluate our method on three public benchmarks: RWARE [Christianos et al., 2020], ProcTHOR [Deitke et al., 2022], and ALFWorld [Shridhar et al., 2021]. The results show that our method achieves significant improvements over existing baseline methods in both effectiveness and efficiency. **Our main contributions are summarized as follows:**

- **Margin-aware Direct Preference Optimization** dynamically scales preference margins based on the severity of physical errors. This addresses the severity limitation of standard DPO methods and encourages LLMs to better follow critical physical constraints.
- **Verifier-Guided Search Algorithm** performs hard symbolic pruning during inference, supporting efficient real-time embodied planning.
- **App-200 Embodied Planning Benchmark** is the first execution-verified dataset with the step-level error severity labels, enabling a fine-grained analysis of failure modes.
- **Comprehensive Evaluation** on four public datasets, including success rate and latency comparisons, experiments with different training and inference settings, tests under incomplete verifier rules, and comparisons of different priors under the same search process, validates the effectiveness and efficiency of MAPS.

2 Related Work

Embodied Planning Techniques Driven by Large Language Models. Advances in large language models (LLMs) have propelled the rapid development of embodied planning, making it a core technological solution for robotic agents to perform cross-modal perception, logical reasoning, and closed-loop action execution. Early research focused primarily on zero-shot text instruction decomposition methods [Huang et al., 2022], followed by usability-aware systems such as SayCan [ichter et al., 2023] and multimodal planners such as PaLM-E [Driess et al., 2023]. To bridge the gap between general language capabilities and specific embodied tasks, subsequent research has widely adopted supervised fine-tuning and imitation learning paradigms, with representative frameworks including AgentTuning [Zeng et al., 2024] and KNOWAGENT [Zhu et al., 2025]. Simultaneously, research has further expanded the boundaries of model capabilities through process knowledge graphs [Zhou et al., 2025], implicit planning coupling [Huang et al., 2025], and active knowledge acquisition [Liu et al., 2025]. At the model alignment level, preference learning, centered on Direct Preference Optimization (DPO) [Rafailov et al., 2023], has become the standard paradigm for language models to adapt to complex embodied tasks. ODPO extends DPO by setting an offset according to the reward difference between preferred and dispreferred responses, requiring a larger likelihood gap when the preference is stronger [Amini et al., 2024]. Our mDPO instead uses verifier judgments of planning error severity to set a bounded margin and increase the weight of examples with more serious errors. Existing research continuously optimizes the logical reasoning and task alignment capabilities of models through methods such as stepwise rewards [Lai et al., 2024], search feedback [Wang et al., 2024], automatic discovery of preference optimization variants [Lu et al., 2024], and infinite preference fusion [Gu et al., 2025]. To overcome the inherent limitations of single autoregressive decoding, search-based strategies such as ToT (Full Tree) [Yao et al., 2023a] and RAP (Programming Inference) [Hao et al., 2023] endow large language models with long-term, deliberate reasoning capabilities. Furthermore, to bridge the gap between continuous language generation and discrete physical constraints, neurosymbolic methods such as LLM+P [Liu et al., 2023] and LLM-Modulo frameworks [Kambhampati et al., 2024] combine approximate heuristics of large language models with precise external solvers. The core algorithms of these search-based planners largely originate from UCT and Monte Carlo Tree Search (MCTS) frameworks [Kocsis and Szepesvári, 2006, Coulom, 2006, Browne et al., 2012]. In addition, existing research has constructed evaluation systems for embodied planners in the LLM domain from multiple dimensions, including the physical executability of planning and the robustness of independent planning capabilities.

3 Methodology

3.1 Overview of MAPS Framework

To address several core challenges in existing LLM-based embodied planners, including their tendency to generate plans that appear reasonable in language but violate physical or task constraints during execution, the difficulty of existing preference alignment methods in distinguishing the severity of different planning errors, the high inference latency introduced by traditional tree search in constrained planning spaces, and their reliance on large closed-source models that are difficult to deploy on resource-constrained platforms, we propose an end-to-end unified embodied planning framework that combines mDPO-based training optimization with closed-loop inference-time correction. As shown in Figure 1, the framework consists of two closely connected stages that share the same symbolic verifier V throughout training and inference. This verifier checks the executability of each action in the plan, compliance with physical constraints, and completion of the final task goal. During training, it provides error locations and severity labels; during inference, it prunes invalid branches and validates candidate plans before execution. Its signals are used through different interfaces for data construction, severity labeling, search, and evaluation, as summarized in Appendix Table B.1. In the training stage, Margin-Aware Direct Preference Optimization (mDPO) uses verifier-derived severity labels to learn a more robust and physically grounded planner prior (initial planner before search and repair). In the inference stage, VGS uses the same verifier to remove infeasible branches and repair invalid trajectories, thereby improving both planning reliability and inference efficiency.

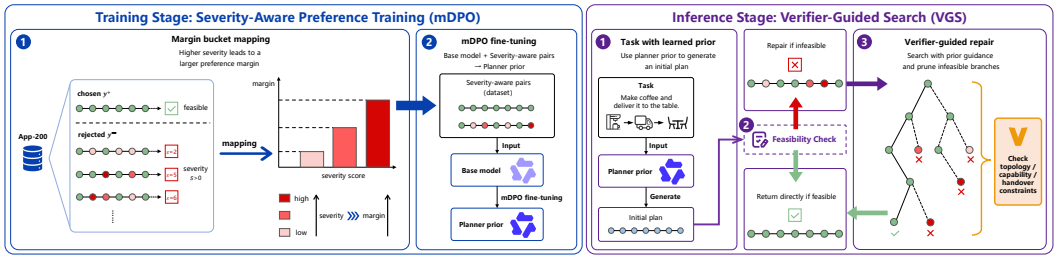


Figure 1: **Overview of Our Unified Embodied Planning Framework. Left: Training stage.** We construct execution-grounded preference data and train the planner with **Margin-aware Direct Preference Optimization (mDPO)**, which uses severity labels to assign different optimization margins and learn a more feasible planner prior. **Right: Inference stage.** We use **VGS** to repair infeasible trajectories and improve both planning reliability and inference efficiency. The verifier **V** in VGS checks action executability, physical constraint satisfaction, and task-goal completion.

3.2 Formalization: Constrained Embodied Planning

Unlike standard text generation tasks, embodied planning must satisfy hard physical and task constraints during execution. We formulate the problem as a constrained embodied planning process, represented by the tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{V}, \mathcal{G} \rangle, \tag{1}$$

where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T}$ is the state transition function, $\mathcal{V}$ is a symbolic verifier, and $\mathcal{G}$ is the terminal goal predicate. Given a task instruction x, the planner defines a policy $\pi_\theta(a_t \mid s_t, x)$. For a generated trajectory $y = [a_1, \ldots, a_T]$ with state sequence $\{s_t\}_{t=0}^{T}$, the verifier evaluates each state-action transition as

$$\mathcal{V}(s_{t-1}, a_t) = \big(\mathcal{V}_{\text{feasible}}(s_{t-1}, a_t), \mathcal{V}_{\text{sev}}(s_{t-1}, a_t)\big), \tag{2}$$

where $\mathcal{V}_{\text{feasible}}(s_{t-1}, a_t) \in \{0, 1\}$ indicates whether the action is executable under the current state, and $\mathcal{V}_{\text{sev}}(s_{t-1}, a_t) \in \mathbb{R}^+$ assigns a nonnegative severity penalty to the transition. The executable success of the entire trajectory is defined as

$$\texttt{exec}(y) = \Big(\bigwedge_{t=1}^{T} \mathcal{V}_{\text{feasible}}(s_{t-1}, a_t) \Big) \wedge \mathcal{G}(s_T), \tag{3}$$

which means that every action in the trajectory must be feasible and the final state must satisfy the task goal. To support severity-aware preference learning, we further define the trajectory penalty as

$$\texttt{penalty}(y) = \sum_{t=1}^{T} \mathcal{V}_{\text{sev}}(s_{t-1}, a_t) + \lambda_g\big(1 - \mathcal{G}(s_T)\big), \tag{4}$$

where $\lambda_g > 0$ is the penalty for terminal goal failure. This trajectory penalty is later used to measure the severity gap between the chosen plan y^+ (the preferred/valid reference plan) and the rejected plan y^- (the dispreferred or invalid candidate plan) in mDPO, while the verifier's feasibility check is reused during inference to prune invalid branches in VGS.

3.3 Training Stage Planner Prior via Margin-Aware Direct Preference Optimization (mDPO)

Motivation: Standard Direct Preference Optimization (DPO) is unable to distinguish error severity: it treats all negative plans as the same type of failure and therefore applies similar optimization pressure to very different errors, such as formatting errors and severe physical constraint violations. This makes it difficult for the model to learn hard physical constraints and build a robust planner prior that better follows physical constraints. To address this, we construct a four-stage closed-loop data pipeline to generate pairwise preference data (y^+, y^-) based on real execution outcomes (see Section 3.6 and Appendix B for full details).

Margin-aware DPO (mDPO): To align preference learning with execution risk, we assign each rejected plan y_i^- a severity score computed by the verifier based on the trajectory-level penalty:

$$\text{sev}_i = \texttt{penalty}(y_i^-) - \texttt{penalty}(y_i^+). \tag{5}$$

We then map this score into a margin bucket for each sample:

$$m_i = \mathcal{B}(\text{sev}_i). \tag{6}$$

where larger margins correspond to more severe failures. The standard DPO preference logit is defined as

$$r_i = \Big(\log \frac{\pi_\theta(y_i^+|x_i)}{\pi_{\text{ref}}(y_i^+|x_i)} - \log \frac{\pi_\theta(y_i^-|x_i)}{\pi_{\text{ref}}(y_i^-|x_i)}\Big). \tag{7}$$

We formulate mDPO as:

$$\mathcal{L}_{\text{mDPO}}(\theta) = -\mathbb{E}_{(x_i, y_i^+, y_i^-)}\Big[w_i \log \sigma\big(\beta(r_i - \alpha m_i)\big)\Big], \tag{8}$$

The sample weight is defined as

$$w_i = 1 + \lambda m_i. \tag{9}$$

The margin term αm_i makes the model keep a larger gap between the chosen plan and more severe failures, while the sample weight w_i can further emphasize more serious violations during training.

3.4 Inference Stage Correction via VGS

Motivation: Even with a stronger planner prior, autoregressive decoding can still produce invalid plans in tightly constrained embodied environments. Errors that remain after training may accumulate across multiple steps and eventually cause the whole trajectory to fail. To address this, we introduce VGS as the inference stage correction module. In our implementation, VGS uses Monte Carlo Tree Search (MCTS) as the underlying search framework [Coulom, 2006, Silver et al., 2017]. During inference, VGS uses the symbolic verifier to check candidate transitions, remove invalid branches, and repair infeasible trajectories.

Search Logic: Each search node represents the current state of the task together with its action history. The symbolic verifier first defines the set of feasible actions under the current state:

$$\mathcal{A}_{\text{valid}}(s) = \{a \in \mathcal{A} \mid \mathcal{V}_{\text{feasible}}(s, a) = 1\}. \tag{10}$$

This hard pruning step removes invalid transitions before they enter the search tree. For tree selection, we use the standard Upper Confidence Bound (UCB) rule:

$$\text{UCB}(n) = \frac{W(n)}{N(n)} + c_{\text{uct}}\sqrt{\frac{\log N(p(n))}{N(n)}}, \tag{11}$$

where $W(n)$ is the accumulated value of node n, $N(n)$ is its visit count, and $p(n)$ denotes its parent node. During expansion and rollout, VGS uses a fixed heuristic to score the current state and the initial plan to guide the search. The next action is selected according to

$$a^\star = \arg\max_{a \in \mathcal{A}_{\text{valid}}(s)} \Big(\texttt{score}(s') + \texttt{prior}(a, d) \Big), \qquad s' = \mathcal{T}(s, a), \tag{12}$$

where, $\texttt{score}(s')$ and $\texttt{prior}(a, d)$ are fixed heuristics defined by explicit rules. The verifier first removes infeasible actions, and the remaining actions are ranked by task progress, execution cost, and consistency with the initial plan.

Finally, VGS returns the repaired trajectory found during search, which has the highest score among all candidates. If no valid repair is found, the system falls back to the greedy plan if it is executable; otherwise, it returns `Failure`. The detailed procedure is given in Algorithm B.2.

3.5 Gradient Analysis of mDPO

We analyze the gradient of mDPO to explain why it emphasizes severe failures. For one preference pair, the loss is

$$\mathcal{L}_i = -w_i \log \sigma\big(\beta(r_i - \alpha m_i)\big), \tag{13}$$

where m_i is the severity margin and $w_i = 1 + \lambda m_i$ with $\lambda \geq 0$. Its gradient is

$$\nabla_\theta \mathcal{L}_i = -w_i \beta \Big(1 - \sigma\big(\beta(r_i - \alpha m_i)\big)\Big) \nabla_\theta r_i. \tag{14}$$

Let

$$G_i = w_i \beta \Big(1 - \sigma\big(\beta(r_i - \alpha m_i)\big)\Big), \tag{15}$$

so $\nabla_\theta \mathcal{L}_i = -G_i \nabla_\theta r_i$. For fixed r_i, if $\alpha > 0$ and $\lambda \geq 0$, G_i increases with m_i: a larger m_i lowers $(r_i - \alpha m_i)$ and increases $1 - \sigma(\cdot)$, and w_i also grows with m_i. Under the same preference gap r_i, more severe rejected plans receive larger gradient updates than mild ones. Thus, mDPO pushes the model more strongly away from severe failures and improves sensitivity to errors that can directly break execution. Additional boundary, gradient, and curvature results are in Appendix A.

3.6 Four-Stage Data Construction Pipeline

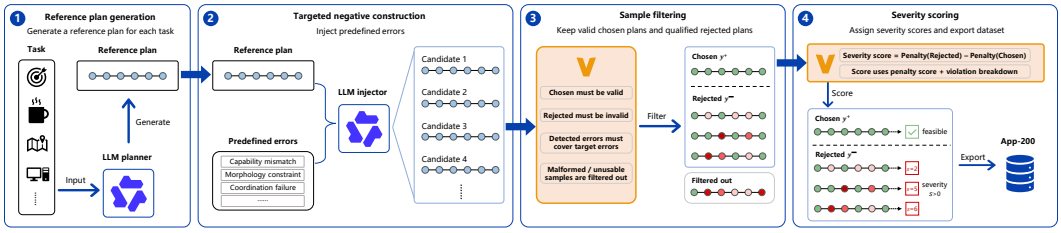


Figure 2: **Four-Stage Automatic Data Construction Pipeline.** Stage 1 generates one executable reference plan per task. Stage 2 constructs targeted negative candidates by injecting predefined error types. Stage 3 filters pairs and keeps only usable chosen-rejected samples. Stage 4 assigns severity scores and exports the final App-200 training set.

We build mDPO training data with the four-stage pipeline in Figure 2. Each sample contains a chosen plan y^+ (the preferred valid reference plan) and a rejected plan y^- (the negative candidate plan). **Stage 1: Reference plan generation.** For each task, we generate one executable reference plan and keep it as the chosen plan y^+. **Stage 2: Targeted negative construction.** Starting from the reference plan, we inject predefined error types to create negative candidates, such as capability mismatch, morphology constraint violation, and coordination failure. **Stage 3: Sample filtering.** We keep only high quality pairs. A kept pair must satisfy: (1) y^+ is valid, (2) y^- is invalid, and (3) the detected error in y^- matches the target injected error type. Malformed outputs are removed. **Stage 4: Severity scoring and export.** For each kept pair, we assign a severity score

$$s = \text{Penalty}(y^-) - \text{Penalty}(y^+),$$

where $\text{Penalty}(\cdot)$ is the trajectory penalty based on the verifier. The score uses both the penalty value and the types of violation error. We then export the prompt, y^+, y^-, the feedback of the verifier, and s to form the final set of App-200 training.

4 Experiments

We evaluate MAPS from four aspects: overall performance, VGS effectiveness, mDPO effectiveness, and runtime efficiency. We use the following research questions:

- **RQ1 (Overall Performance):** Does the full MAPS system outperform strong baselines across different embodied planning benchmarks?
- **RQ2 (Effectiveness Analysis on VGS):** How much does VGS improve final success?
- **RQ3 (Effectiveness Analysis on mDPO):** How much does mDPO improve planner quality before VGS repair?
- **RQ4 (Efficiency Comparison):** What is the efficiency of MAPS during the inference phase?

4.1 Datasets and Experimental Settings

Datasets. We evaluate MAPS on four embodied planning benchmarks using one shared executable evaluation protocol. We introduce **App-200** in this work for multi-robot delivery planning under hard physical constraints. It contains 200 execution-verified tasks, with about 12 steps per task on average. The tasks differ in pickup source, handover requirement, and handover location, so models cannot solve them by relying on only a few repeated route patterns. App-200 also provides step-level error severity labels, which let us analyze not only whether a task fails, but also what kind of failure it contains. Appendix Table C.2 summarizes the task composition and basic statistics of App-200. We also test generalization on three public benchmarks: **RWARE** [Christianos et al., 2020], which focuses on constrained multi-agent routing; **ProcTHOR** [Deitke et al., 2022], which focuses on embodied planning in diverse 3D indoor scenes; and **ALFWorld** [Shridhar et al., 2021], which evaluates interactive household decision making from language instructions.

Baselines and Metrics. We compare our system with several baselines under the same evaluation protocol. The major baselines include **ReAct-style direct planning** [Yao et al., 2023b, Yang et al., 2024], **Reflexion-style retry** [Shinn et al., 2023, Pan et al., 2024], **ToT-style search [Yao et al., 2023a]**, and several planner training methods, including Base LLM, **SFT** [Zeng et al., 2024], **Vanilla DPO** [Rafailov et al., 2023], **SimPO** [Meng et al., 2024], **ORPO** [Hong et al., 2024]. The details of baselines are shown in Appendix G.

Our main metrics are **exact execution success rate** and **average inference latency per task**. In the ablation studies, we also report how often the planner makes format errors (**Format Error@100**), how often it causes severe morphology violations (**Severe Morph. Viol.**), and how often it reaches logical completion (**Logical Comp.**).

Experimental Settings. All methods use the same executable rules, prompt format, success criterion, task splits, and fixed task IDs. The implementation details including hyperparameters and prompts are described in Appendix E and Appendix F.

4.2 Main Results (RQ1 and RQ2)

RQ1: Overall Performance. Table 1 answers RQ1 by comparing complete systems on four embodied planning benchmarks. MAPS gives the best results on App-200, ProcTHOR, and ALFWorld, and is also close to ToT on RWARE (0.6350 vs. 0.6392). This shows that MAPS works well across different embodied environments, rather than only on the new App-200 benchmark. Appendix Figure C.1 gives a fuller flow view of how tasks move from the planner's initial output to repaired or unresolved final outcomes on each dataset, and shows that VGS repairs many initially failed cases across all four benchmarks. However, these overall system results do not show how much of the improvement comes from better planner training and how much comes from repair during inference. We examine that next with a controlled ablation study.

RQ2: Effectiveness Analysis on VGS To measure the contribution of VGS, we compare methods under matched settings. Figure 3(a1)–(a2) compares ReAct-style direct planning, Reflexion-style retry, and VGS on App-200 and ProcTHOR, while keeping the planner prior fixed. Across all three planner priors (SFT, Vanilla DPO, and mDPO), VGS achieves the highest final success. This result shows that VGS consistently improves outcomes after the first generated plan. Figure 3(a3) gives

Table 1: System performance on exact execution success rate under the Qwen2.5-7B backbone. The results are mean ± std over 3 runs. Best results are in **bold**.

Method	App-200	RWARE (pooled 400)	ProcTHOR	ALFWorld
ReAct-style one-shot (Qwen base)	0.8033 ± 0.0202	0.0825 ± 0.0050	0.7167 ± 0.0153	0.6367 ± 0.0058
Reflexion-style retry (Qwen base)	0.8700 ± 0.0150	0.5817 ± 0.0076	0.7600 ± 0.0436	0.6833 ± 0.0252
ToT-style search (Qwen base)	0.9183 ± 0.0104	$\mathbf{0.6392 \pm 0.0063}$	0.9100 ± 0.0100	0.8867 ± 0.0153
Seq2Seq-style one-shot (Qwen SFT)	0.8133 ± 0.0126	0.0233 ± 0.0052	0.7567 ± 0.0252	0.6433 ± 0.0404
Vanilla DPO-style one-shot (Qwen DPO)	0.8033 ± 0.0058	0.0858 ± 0.0072	0.7400 ± 0.0265	0.6433 ± 0.0058
Ours (mDPO + VGS)	$\mathbf{0.9650 \pm 0.0100}$	0.6350 ± 0.0075	$\mathbf{0.9367 \pm 0.0153}$	$\mathbf{0.9367 \pm 0.0208}$

a combined view of the repair process. It shows how tasks move from the first result (success or failure), to retry results, and then to final results after VGS. The ssuccess rate shows that VGS fixes many failed cases, not only a small number of easy ones.

Taken together, these matched comparisons show that VGS is the main reason for the final system gain under hard constraints. Appendix Table C.4 gives the exact scores, and Appendix Table C.3 gives the paired significance test results.

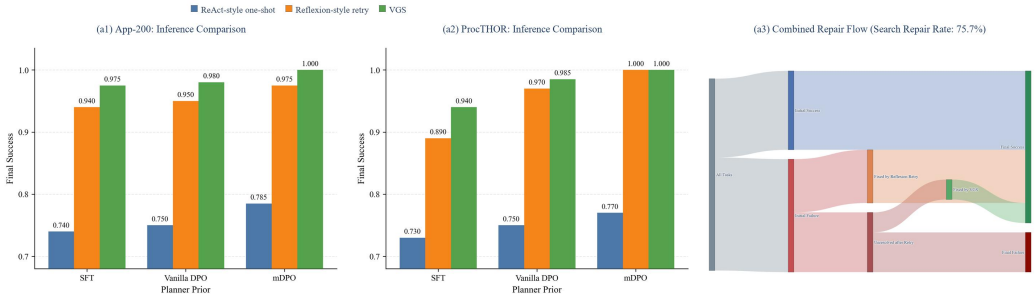


Figure 3: **Component ablation for the effectiveness of VGS.** In subplot (a1) and (a2), we compare ReAct-style direct planning, Reflexion-style retry, and VGS under three planner training methods. In (a3), we demonstrate how VGS assesses the feasibility of candidate plans and iteratively refines them during the inference stage.

4.3 Effectiveness Analysis on mDPO (RQ3)

RQ3: How much does mDPO improve planner quality before VGS ? Figure 4 answers RQ3 from two views. In subplot (a), we keep VGS fixed and compare different planner priors. The mean final success increases from 0.958 (SFT) and 0.982 (Vanilla DPO) to 1.000 (mDPO). This shows that planner quality still affects final outcomes even when the VGS is the same. In subplot (b), we compare planner quality under direct planning (one pass planning without VGS) on Qwen2.5-7B and Llama-3.1-8B. We use four metrics: Exec Success (exact execution success on the 100 tasks planner quality set), Logic (task logic completion rate), Format Robustness (one minus format error rate), and Morph Safety (one minus severe morphology violation rate). Across both backbones, mDPO gives the strongest overall profile. Compared with Vanilla DPO, mDPO improves Qwen2.5-7B Exec Success from 0.3367 to 0.3767 and Logic from 0.3900 to 0.4233; on Llama-3.1-8B, it improves Exec Success from 0.3267 to 0.3500 and Logic from 0.3433 to 0.3733. These results show that mDPO mainly helps by improving the initial planner and reducing severe failures before VGS.

To further test whether mDPO improves the planner itself, we also evaluate generalization under two splits: in-distribution (ID, train and test from the same building) and out-of-distribution (OOD, test from unseen buildings). Table 2 shows that on ID, SFT, Vanilla DPO, and mDPO are close (0.671–0.684). On OOD, all methods drop, which means cross-building transfer is still hard. However, mDPO keeps the best OOD result (0.310), clearly above SFT (0.181) and Vanilla DPO (0.140). This result is consistent with Figure 4: mDPO gives a stronger planner prior and better robustness in harder transfer settings.

4.4 Efficiency Comparison (RQ4)

RQ4: How effectively does VGS repair failed plans, and how much extra inference time does it add? Table 3 gives the main answer to RQ4. On both App-200 and ProcTHOR, VGS reaches much

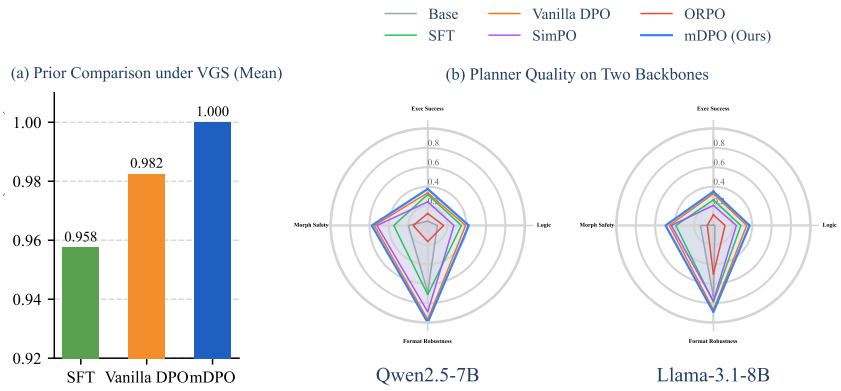


Figure 4: **mDPO effectiveness on planner quality (RQ3).** In subplot (a), we keep VGS fixed and compare SFT, Vanilla DPO, and mDPO. In subplot (b), we compare Base, SFT, Vanilla DPO, SimPO, ORPO, and mDPO on Qwen2.5-7B and Llama-3.1-8B using four planner quality metrics. Larger is better on all axes.

Table 2: Generalization of planner priors under ID and OOD splits. Methods are similar on ID, while mDPO is strongest on OOD.

Evaluation Split	Method	Exact Success ↑
ID (Same building)	SFT	0.678
	Vanilla DPO	0.671
	mDPO (Ours)	**0.684**
OOD (Cross building)	SFT	0.181
	Vanilla DPO	0.140
	mDPO (Ours)	**0.310**

higher final success than ReAct-style direct planning. Compared with Reflexion-style retry, it also raises the average final success from 0.975 to 0.990, while reducing the average end-to-end runtime from 12.814s to 10.969s. On both datasets, the extra time beyond LLM decoding is only 0.045s per task, which is less than 0.5% of the total runtime. This shows that VGS can repair many failed initial plans while adding very little extra time. We next look more closely at which failed cases VGS fixes and why the extra time stays small.

Appendix Figure C.1 shows more clearly how tasks move from the planner's initial output to repaired or unresolved final outcomes across the four benchmarks. Appendix Table C.3 further shows that these repair gains are reliable on the same matched tasks. Appendix Table C.6 shows where the extra time comes from, including symbolic rule checking, VGS search control, and extra decoding rounds in retry. Appendix Table C.5 also shows that even when the VGS search budget is increased, the added search cost remains at the millisecond level. Finally, Appendix D provides representative repair traces and safe abstention cases.

Table 3: **End-to-end latency comparison on App-200 and ProcTHOR.** 'Total Time' is the average end-to-end runtime per task. 'LLM Gen.' is the time spent on model token generation. 'Extra Time' is the remaining time outside model generation, mainly rule checking and VGS control. Most inference time comes from model generation, and for VGS the extra time remains below 0.5% of total runtime on both datasets.

Dataset	Method	Final Success	Total Time (s) ↓	LLM Gen. (s) ↓	Extra Time (s) ↓
App-200	ReAct-style one-shot	0.820	11.596	11.591	0.005
App-200	Reflexion-style retry	0.960	13.405	13.400	0.005
App-200	VGS	0.980	11.628	11.583	0.045
ProcTHOR	ReAct-style one-shot	0.820	10.287	10.282	0.005
ProcTHOR	Reflexion-style retry	0.990	12.223	12.217	0.006
ProcTHOR	VGS	1.000	10.309	10.264	0.045

5 Conclusion

This paper presents MAPS, a unified framework for embodied planning that combines Margin-Aware Direct Preference Optimization (mDPO) during training with VGS during inference. MAPS uses the same symbolic verifier in data construction, preference learning, and plan repair, so both training and inference follow the same execution rules. Across four embodied planning benchmarks, MAPS achieves an average success rate of 86.84%, outperforming the best competing baseline by 2.98 percentage points. Our experiments further show that the two components play different roles: VGS provides the largest final gains by repairing invalid plans, while mDPO improves the planner's initial output and reduces severe failures before repair. We also find that most inference time comes from plan generation by the model, while rule checking and search add only a small amount of extra time. Overall, these results show that strengthening the planner first and then repairing the remaining failures with a verifier is an effective way to improve embodied planning under hard constraints.

6 Limitations

Our evaluation still relies mainly on symbolic execution rather than real hardware, so it does not capture sensing noise, control error, or execution uncertainty. Transfer to harder out-of-distribution settings, such as cross-building tasks, also remains challenging. These limitations also raise safety considerations. MAPS can make embodied planning more reliable in tasks with clear constraints by reducing serious planning errors and fixing invalid plans before execution, which is useful in settings such as service robots, warehouse systems, and assistive robots, where planning mistakes may waste time, cause task failure, or lead to unsafe behavior. However, these results do not mean the system is fully safe for real-world autonomous use. First, a stronger embodied planner could still be misused in harmful settings. Second, the framework depends on the symbolic verifier being correct and complete; if important rules are missing or wrong, the system may still accept unsafe plans. Third, our experiments are based mainly on symbolic execution rather than physical robots, so they do not reflect sensing noise, control errors, or hardware uncertainty. MAPS should therefore be viewed as a tool for constrained planning, not as a complete safety guarantee.

Future work will extend the framework to physical robots, harder coordination settings, and broader embodied systems.

Acknowledgments and Disclosure of Funding

Lei Chen's work is partially supported by National Key Research and Development Program of China Grant No. 2023YFF0725100, National Science Foundation of China (No. U22B2060), Guangdong-Hong Kong Technology Innovation Joint Funding Scheme Project No. 2024A0505040012, the Hong Kong RGC GRF Project 16213620, RIF Project R6020-19, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Key Areas Special Project of Guangdong Provincial Universities 2024ZDZX1006, Guangdong Province Science and Technology Plan Project 2023A0505030011, Guangzhou municipality big data intelligence key lab, 2023A03J0012, Hong Kong ITC ITF grants MHX/078/21 and PRP/004/22FX, Hong Kong ITC TC-SKLCRCC26EG01, Zhujiang scholar program 2021JC02X170, Microsoft Research Asia Collaborative Research Grant, HKUST-Webank joint research lab, 2025 HKUST Shenzhen-Hong Kong Collaborative Innovation Institute Green Sustainability Special Fund from Shui On Xintiandi and the InnoSpace GBA, and HKUST(GZ) - CMCC(Guangzhou Branch) Metaverse Joint Innovation Lab under Grant No. P00659.

Competing Interests Disclosure

The authors declare that they have no competing interests.

References

Eitan Altman. *Constrained Markov decision processes*. Routledge, 1999.

Afra Amini, Tim Vieira, and Ryan Cotterell. Direct preference optimization with an offset. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, 2024. URL https://openreview.net/forum?id=mA28B_PCL9.

Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43, 2012.

Wonje Choi, Jooyoung Kim, and Honguk Woo. NeSyPr: Neurosymbolic proceduralization for efficient embodied reasoning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=a8sJEH4Cjb.

Filippos Christianos, Lukas Schäfer, and Stefano Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 10707–10717, 2020.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Proceedings of the 5th International Conference on Computers and Games*, pages 72–83, 2006.

Matt Deitke, Eli VanderBilt, Alvaro Herrasti, Luca Weihs, Kiana Ehsani, Jordi Salvador, Winson Han, Eric Kolve, Aniruddha Kembhavi, and Roozbeh Mottaghi. ProcTHOR: Large-scale embodied AI using procedural generation. In *Advances in Neural Information Processing Systems*, volume 35, pages 5982–5994, 2022.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-E: An embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pages 8469–8488, 2023.

Yanggan Gu, Yuanyi Wang, Zhaoyi Yan, Yiming Zhang, Qi Zhou, Fei Wu, and Hongxia Yang. InfiFPO: Implicit model fusion via preference optimization in large language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=ZwBtDbuzjY.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. Reasoning with language model is planning with world model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173, 2023.

Jiwoo Hong, Noah Lee, and James Thorne. ORPO: Monolithic preference optimization without reference model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11170–11189, 2024.

Chi-Pin Huang, Yueh-Hua Wu, Min-Hung Chen, Yu-Chiang Frank Wang, and Fu-En Yang. ThinkAct: Vision-language-action reasoning via reinforced visual latent planning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=72UR53jN7T.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pages 9118–9147, 2022.

brian ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter

Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as I can, not as I say: Grounding language in robotic affordances. In *Proceedings of The 6th Conference on Robot Learning*, volume 205, pages 287–318, 2023.

Daniel Kahneman. *Thinking, fast and slow*. Macmillan, 2011.

Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. Position: LLMs can't plan, but can help planning in LLM-modulo frameworks. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pages 22895–22907, 2024.

Minbeom Kim, Hwanhee Lee, Kang Min Yoo, Joonsuk Park, Hwaran Lee, and Kyomin Jung. Critic-guided decoding for controlled text generation. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 4598–4612, 2023.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. Step-DPO: Step-wise preference optimization for long-chain reasoning of LLMs. *arXiv preprint arXiv:2406.18629*, 2024.

Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. LLM+P: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023.

Xiaotian Liu, Ali Pesaranghader, Jaehong Kim, Tanmana Sadhu, Hyejeong Jeon, and Scott Sanner. ActiveVOO: Value of observation guided active knowledge acquisition for open-world embodied lifted regression planning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=cZVYswQQMt.

Chris Lu, Samuel Holt, Claudio Fanconi, Alex J. Chan, Jakob Foerster, Mihaela van der Schaar, and Robert Tjarko Lange. Discovering preference optimization algorithms with and for large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 86528–86573, 2024.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=3Tzcot1LKb.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. Automatically correcting large language models: Surveying the landscape of diverse automated correction strategies. *Transactions of the Association for Computational Linguistics*, 12:484–506, 2024.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Cote, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0IOX0YcCdTn.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, 2017.

Charlie Victor Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling LLM test-time compute optimally can be more effective than scaling parameters for reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=4FWAwZtd2n.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. Chain of thoughtlessness? An analysis of CoT in planning. In *Advances in Neural Information Processing Systems*, volume 37, pages 29106–29141, 2024.

Karthik Valmeekam, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. Large language models still can't plan (a benchmark for LLMs on planning and reasoning about change). In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. URL https://openreview.net/forum?id=wUU-7XTL5XO.

Karthik Valmeekam, Matthew Marquez, Alberto Olmo, Sarath Sreedharan, and Subbarao Kambhampati. PlanBench: An extensible benchmark for evaluating large language models on planning and reasoning about change. In *Advances in Neural Information Processing Systems*, volume 36, pages 38975–38987, 2023.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18:186345, 2023.

Tianlong Wang, Junzhe Chen, Xueting Han, and Jing Bai. CPL: Critical plan step learning boosts LLM generalization in reasoning tasks. *arXiv preprint arXiv:2409.08642*, 2024.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huang, and Tao Gui. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68:121101, 2025.

Jian Xie, Kexun Zhang, Jiangjie Chen, Siyu Yuan, Kai Zhang, Yikai Zhang, Lei Li, and Yanghua Xiao. Revealing the barriers of language agents in planning. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1872–1888, 2025.

Zonghan Yang, Peng Li, Ming Yan, Ji Zhang, Fei Huang, and Yang Liu. ReAct meets ActRe: When language agents enjoy training data autonomy. *arXiv preprint arXiv:2403.14589*, 2024.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822, 2023a.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=WE_vluYUL-X.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 3053–3077, 2024.

Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. In *RSS 2023 Workshop on Learning for Task and Motion Planning*, 2023. URL https://openreview.net/forum?id=tED747HURfX.

Jiacong Zhou, Jiaxu Miao, Xianyun Wang, and Jun Yu. Enhancing LLM planning for robotics manipulation through hierarchical procedural knowledge graphs. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=8LO0vLRXpz.

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, Huajun Chen, and Ningyu Zhang. KnowAgent: Knowledge-augmented planning for LLM-based agents. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3709–3732, 2025.

Appendix

A Additional Properties and Gradient Analysis of mDPO

This section provides additional properties and gradient analysis for the Margin-Aware Direct Preference Optimization (mDPO) objective introduced in Section 3.3. We expand the discussion in Section 3.5 and show how the severity margin changes the preference boundary and the gradient signal.

Proposition A.1 (Severity Shifted Preference Boundary). For a preference pair i, let $z_i = r_i - \alpha m_i$, where r_i is the standard DPO preference logit, $\alpha \geq 0$, and $m_i \geq 0$ is the severity margin. Then the mDPO objective prefers the chosen plan y_i^+ over the rejected plan y_i^- when $r_i > \alpha m_i$. Compared with vanilla DPO, mDPO shifts the preference boundary by a severity dependent margin αm_i.

Proof. For a single preference pair, the mDPO loss is

$$\mathcal{L}_i = -w_i \log \sigma\big(\beta z_i\big).$$

where $w_i = 1 + \lambda m_i > 0$ and $\sigma(\cdot)$ is the sigmoid function. Since w_i is a positive scalar and $\sigma(\cdot)$ is strictly increasing, minimizing $\mathcal{L}_i$ is equivalent to maximizing z_i. Therefore, the chosen plan is preferred whenever

$$z_i > 0.$$

Substituting $z_i = r_i - \alpha m_i$, we obtain

$$r_i - \alpha m_i > 0 \quad \Longleftrightarrow \quad r_i > \alpha m_i.$$

Thus, larger severity margins require a larger preference gap before the chosen plan is preferred. In this way, mDPO keeps severe negative samples farther from valid plans and makes the model less likely to treat them as similar. ■

Proposition A.2 (Gradient and Curvature Bounds). For the mDPO loss $\mathcal{L}_i = -w_i \log \sigma\big(\beta(r_i - \alpha m_i)\big)$ with $w_i = 1 + \lambda m_i$ and $\lambda \geq 0$, the gradient and curvature with respect to the preference logit r_i are bounded as follows:

$$\frac{\partial \mathcal{L}_i}{\partial r_i} = -w_i \beta \Big(1 - \sigma\big(\beta(r_i - \alpha m_i)\big)\Big), \tag{A.1}$$

$$\left|\frac{\partial \mathcal{L}_i}{\partial r_i}\right| \leq \beta(1 + \lambda m_{\max}), \tag{A.2}$$

$$\frac{\partial^2 \mathcal{L}_i}{\partial r_i^2} = w_i \beta^2 \sigma\big(\beta(r_i - \alpha m_i)\big)\Big(1 - \sigma\big(\beta(r_i - \alpha m_i)\big)\Big), \tag{A.3}$$

$$\frac{\partial^2 \mathcal{L}_i}{\partial r_i^2} \leq \frac{\beta^2}{4}(1 + \lambda m_{\max}), \tag{A.4}$$

where $m_{\max}$ is the largest severity margin.

Proof. Let $u_i = \beta(r_i - \alpha m_i)$. Then

$$\mathcal{L}_i = -w_i \log \sigma(u_i). \tag{A.5}$$

Using $\frac{d}{du}\sigma(u) = \sigma(u)(1 - \sigma(u))$, we have

$$\frac{\partial \mathcal{L}_i}{\partial u_i} = -w_i \frac{1}{\sigma(u_i)} \sigma(u_i)(1 - \sigma(u_i)) = -w_i(1 - \sigma(u_i)). \tag{A.6}$$

Since

$$\frac{\partial u_i}{\partial r_i} = \beta, \tag{A.7}$$

the chain rule gives

$$\frac{\partial \mathcal{L}_i}{\partial r_i} = -w_i \beta \Big(1 - \sigma\big(\beta(r_i - \alpha m_i)\big)\Big). \tag{A.8}$$

Because $\sigma(\cdot) \in (0,1)$ and $w_i \le 1 + \lambda m_{\max}$, the absolute gradient is bounded by $\beta(1 + \lambda m_{\max})$.

For the second derivative,

$$\frac{\partial^2 \mathcal{L}_i}{\partial r_i^2} = w_i \beta^2 \sigma\big(\beta(r_i - \alpha m_i)\big)\Big(1 - \sigma\big(\beta(r_i - \alpha m_i)\big)\Big). \tag{A.9}$$

Since $\sigma(u)(1-\sigma(u)) \le \frac{1}{4}$ for all u, we obtain

$$\frac{\partial^2 \mathcal{L}_i}{\partial r_i^2} \le \frac{\beta^2}{4}(1 + \lambda m_{\max}). \tag{A.10}$$

■

Proposition A.3 (Parameter-Level Gradient Mapping). The gradient of the mDPO objective for the model parameters makes the model increase the probability of the chosen plan and decrease the probability of the rejected plan, with a stronger effect on more severe errors:

$$\nabla_\theta \mathcal{L}_{\mathrm{mDPO}}(\theta) = -\mathbb{E}_{(x_i, y_i^+, y_i^-)}\Big[\beta w_i \big(1 - \sigma(\beta(r_i - \alpha m_i))\big)\big(\nabla_\theta \log \pi_\theta(y_i^+|x_i) - \nabla_\theta \log \pi_\theta(y_i^-|x_i)\big)\Big]. \tag{A.11}$$

Proof. Using Proposition A.2, we have

$$\frac{\partial \mathcal{L}_i}{\partial r_i} = -w_i \beta\Big(1 - \sigma\big(\beta(r_i - \alpha m_i)\big)\Big). \tag{A.12}$$

By the chain rule,

$$\nabla_\theta \mathcal{L}_i = \frac{\partial \mathcal{L}_i}{\partial r_i} \nabla_\theta r_i. \tag{A.13}$$

Since the reference policy π_{ref} is fixed, its gradient is zero. Therefore,

$$\nabla_\theta r_i = \nabla_\theta \log \pi_\theta(y_i^+|x_i) - \nabla_\theta \log \pi_\theta(y_i^-|x_i). \tag{A.14}$$

Proposition A.3 shows that mDPO strengthens the update on more severe failures. When the model already prefers the chosen plan, the update in vanilla DPO becomes small, but mDPO keeps it larger for severe errors because of the margin term. ■

B Algorithmic Details for Reproducibility

For reproducibility, we describe the data construction process and the verifier-guided inference controller in detail below. The same symbolic verifier and task specification are used across training, inference, and evaluation, but different verifier signals serve different purposes.

Table B.1: Different uses of the shared symbolic verifier.

Stage	Signal	Purpose
Data construction	Trajectory validity	Keep valid and invalid plans
mDPO labeling	$\mathcal{V}_{\mathrm{sev}}$	Set severity margins and weights
VGS pruning	$\mathcal{V}_{\mathrm{feasible}}$	Remove invalid transitions
VGS ranking	$\texttt{score}(s') + \texttt{prior}(a, d)$	Rank feasible actions
Evaluation	$\mathcal{V}_{\mathrm{feasible}}$ and $\mathcal{G}(s_T)$	Measure executable success

Detailed Logic for Data Construction (Algorithm B.1). The data pipeline includes four stages. In Stage 1, we sample candidate trajectories from the base planner using a slightly higher decoding temperature, so that the sampled plans include both successful cases and common failures. In Stage 2, the symbolic verifier checks each candidate step by step and returns a validity label and a severity score for each violation. In Stage 3, we keep only pairs with a clear difference in quality: the chosen plan must be executable, while the rejected plan must contain a clear execution error. In Stage 4, we split tasks by environment topology before export, so that the training and evaluation sets do not overlap.

Algorithm B.1 Four-Stage Execution-Grounded Data Construction

Require: Task set $\mathcal{X}$, base planner π_θ, symbolic verifier $\mathcal{V} = (\mathcal{V}_{\text{feasible}}, \mathcal{V}_{\text{sev}})$, generation temperature τ_{gen}, severity threshold γ_{sev}
Ensure: Preference dataset $\mathcal{D}_{\text{train}}$, benchmark split $\mathcal{D}_{\text{eval}}$

Initialize $\mathcal{D}_{\text{train}} \leftarrow \emptyset$, $\mathcal{D}_{\text{eval}} \leftarrow \emptyset$
for each task instruction $x \in \mathcal{X}$ **do**
    Stage 1: Candidate Generation
    Sample N candidate trajectories $\{y_1, \ldots, y_N\} \sim \pi_\theta(\cdot \mid x; \tau_{\text{gen}})$
    Stage 2: Symbolic Validation
    $Y_{\text{valid}} \leftarrow \emptyset$, $Y_{\text{invalid}} \leftarrow \emptyset$
    for each candidate $y_i = [a_1, \ldots, a_T]$ **do**
        Roll out states $s_0, s_1, \ldots, s_T$
        $\text{valid}_i \leftarrow \left(\bigwedge_{t=1}^{T} \mathcal{V}_{\text{feasible}}(s_{t-1}, a_t)\right) \wedge \mathcal{G}(s_T)$
        if $\text{valid}_i = 1$ **then**
            $Y_{\text{valid}} \leftarrow Y_{\text{valid}} \cup \{y_i\}$
        else
            $\text{sev}_i \leftarrow \mathcal{V}_{\text{sev}}(y_i)$
            $Y_{\text{invalid}} \leftarrow Y_{\text{invalid}} \cup \{(y_i, \text{sev}_i)\}$
        end if
    end for
    Stage 3: Preference Filtering
    if $|Y_{\text{valid}}| > 0$ **and** $|Y_{\text{invalid}}| > 0$ **then**
        Choose one executable plan $y^+ \in Y_{\text{valid}}$
        Choose $(y^-, \text{sev}^-) \in Y_{\text{invalid}}$ with the largest severity
        if $\text{sev}^- \geq \gamma_{\text{sev}}$ **then**
            $p \leftarrow (x, y^+, y^-, \text{sev}^-)$
            Stage 4: Split and Export
            if x belongs to the OOD evaluation split **then**
                $\mathcal{D}_{\text{eval}} \leftarrow \mathcal{D}_{\text{eval}} \cup \{p\}$
            else
                $\mathcal{D}_{\text{train}} \leftarrow \mathcal{D}_{\text{train}} \cup \{p\}$
            end if
        end if
    end if
end for
return $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{eval}}$

Detailed Logic for Inference Stage Correction (Algorithm B.2). The inference stage correction module is implemented as VGS. It first decodes a greedy plan from the trained planner. If this plan is executable, it is stored as a fallback candidate. The search stage then runs MCTS under verifier guidance. At each expansion step, the verifier filters out infeasible actions through $\mathcal{V}_{\text{feasible}}$, so the search only continues along valid branches. Tree traversal follows the standard UCB rule, and action expansion uses the planner prior and the state score to focus on promising repairs. After the search ends, the system returns the best valid repaired trajectory found during search. If search does not find a valid repair, the system falls back to the greedy plan if that plan is executable; otherwise, it returns `Failure`. In this way, the search cost is controlled by the budget K, and search is only used to fix failed plans. For reproducibility, the two heuristics in Eq. (12) are defined as

$$\texttt{score}(s') = \Phi_{\text{prog}}(s') - C_{\text{path}}(s') - 0.8|h(s')|, \tag{B.15}$$

$$\texttt{prior}(a, d) = q(a, d) + \min\{0.8,\, 0.2 n_{\tilde{y}}(\text{target}(a))\}, \tag{B.16}$$

where $\Phi_{\text{prog}}(s')$ rewards task progress, $C_{\text{path}}(s')$ is the accumulated execution cost, and $|h(s')|$ is the current plan length. For the initial plan $\tilde{y}$, $n_{\tilde{y}}(t)$ counts occurrences of target t. The term $q(a, d)$ equals 1.2 if $a \in \tilde{y}$ and $a \neq \bar{a}_d$, and 0 otherwise, where $\bar{a}_d$ is the initial plan's action at depth d.

Algorithm B.2 Inference Stage Correction via VGS

Require: Task instruction x, trained planner π_θ, symbolic verifier $\mathcal{V} = (\mathcal{V}_{\text{feasible}}, \mathcal{V}_{\text{sev}})$, search budget K
Ensure: Executable plan $\hat{y}$ or `Failure`

Phase 1: Greedy Prior Decoding
$\hat{y}_{\text{prior}} \leftarrow \text{GreedyDecode}(\pi_\theta, x)$
$\hat{y}_{\text{fallback}} \leftarrow$ `Null`
if $\texttt{exec}(\hat{y}_{\text{prior}}) = 1$ **then**
    $\hat{y}_{\text{fallback}} \leftarrow \hat{y}_{\text{prior}}$
end if
Phase 2: Search Initialization
Initialize root node s_0 from task x
$\hat{y}_{\text{search}} \leftarrow$ `Null`
Phase 3: Verifier Guided Search
for $k = 1$ **to** K **do**
    Traverse the tree from s_0 to a leaf node s using the UCB tree policy
    $\mathcal{A}_{\text{valid}}(s) \leftarrow \{a \sim \pi_\theta(\cdot \mid s) \mid \mathcal{V}_{\text{feasible}}(s, a) = 1\}$
    if $\mathcal{A}_{\text{valid}}(s) = \emptyset$ **then**
        continue
    end if
    Select $a^\star \in \mathcal{A}_{\text{valid}}(s)$ using $\texttt{score}(s') + \texttt{prior}(a, d)$
    Expand child state $s' = \mathcal{T}(s, a^\star)$
    Roll out from s' with π_θ to obtain a candidate trajectory y'
    Compute rollout value $R(y')$ and backpropagate it through the visited path
    if $\texttt{exec}(y') = 1$ **then**
        if $\hat{y}_{\text{search}} =$ `Null` **or** $R(y') > R(\hat{y}_{\text{search}})$ **then**
            $\hat{y}_{\text{search}} \leftarrow y'$
        end if
    end if
end for
Phase 4: Final Resolution
if $\hat{y}_{\text{search}} \neq$ `Null` **then**
    return $\hat{y}_{\text{search}}$
else if $\hat{y}_{\text{fallback}} \neq$ `Null` **then**
    return $\hat{y}_{\text{fallback}}$
else
    return `Failure`
end if

C Full Ablation Tables, Supplementary Visualizations, and Extended Analysis

This appendix provides supplementary visualizations, detailed tables, and additional analysis for the results summarized in Section 4.

C.1 App-200 Benchmark Statistics. Table C.2 summarizes what kinds of tasks are included in App-200. We report the task type, pickup source, handover location, estimated complexity, and whether the robot needs to return, to show that App- 200 contains many different task combinations rather than just a few repeated patterns. Here, *estimated complexity* is a simple task category based on the source, handover, and return requirements, not the exact number of steps in the final executed plan.

Table C.2: App-200 benchmark statistics. Counts and shares are reported over all 200 tasks.

Statistic	Count	Share
Number of tasks	200	100.0%
Task profile: Generic	18	9.0%
Task profile: Source-only	49	24.5%
Task profile: Handover-only	38	19.0%
Task profile: Source+Handover	95	47.5%
Pickup source: Starbucks	64	32.0%
Pickup source: Vending-2F	80	40.0%
Pickup source: Unspecified	56	28.0%
Handover endpoint: No handover	67	33.5%
Handover endpoint: Info Hub 4F	57	28.5%
Handover endpoint: Elevator Lobby 4F	76	38.0%
Estimated complexity: Short (1–3)	45	22.5%
Estimated complexity: Medium (4)	90	45.0%
Estimated complexity: Long (5+)	65	32.5%
Return required	138	69.0%
Return not required	62	31.0%

C.2 Paired Significance Analysis. We use the exact McNemar test on matched task IDs to check whether VGS really solves more of the same tasks, rather than only looking better in the average success rate. In Table C.3, 'VGS only' means the number of tasks that VGS solves but the compared method does not. 'Baseline only' means the number of tasks that the compared method solves but VGS does not. Across App pooled tasks and the three public benchmarks, VGS solves many more such extra tasks than it loses, and the reverse cases are almost zero, with $p < 0.001$ in all comparisons. This supports the claim that the repair gains of VGS are statistically reliable.

Table C.3: Paired significance analysis using the exact McNemar test on matched task IDs.

Setting	Matched tasks	VGS only	Baseline only	p-value
App pooled: VGS vs ReAct-style one-shot	308	67	1	< 0.001
App pooled: VGS vs Reflexion-style retry	308	46	1	< 0.001
RWARE: VGS vs ReAct-style one-shot	400	163	0	< 0.001
ProcTHOR: VGS vs ReAct-style one-shot	100	27	0	< 0.001
ALFWorld: VGS vs ReAct-style one-shot	100	36	0	< 0.001

C.3 Clean Factorial Ablation Across Priors and Inference Modes. To separate the effects of planner prior quality and inference-time repair, we compare SFT, Vanilla DPO, and mDPO under ReAct-style one-shot decoding, Reflexion-style retry, and VGS on the same backbone, the same task sets, and the same verifier. Table C.4 reports the exact numerical results behind Figure 3 in the main text. Across both App-200 and ProcTHOR, VGS gives the highest final success under every planner prior, while mDPO gives the strongest results among the compared planner prior under matched

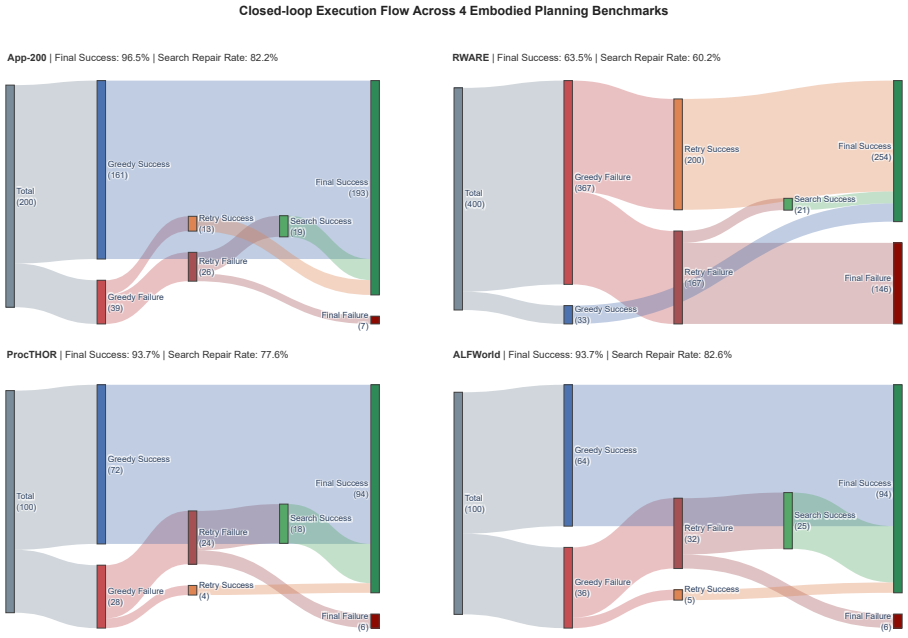


Figure C.1: **Closed loop execution flow across four embodied planning benchmarks.** For each dataset, the Sankey diagram shows how tasks move from total instances to ReAct-style one-shot outcomes, Reflexion-style retry outcomes, VGS repairs, and final outcomes. It shows how many one-shot failures are repaired by VGS and how many cases still remain unresolved.

inference settings.

Table C.4: Clean factorial ablation across priors and inference modes on the same backbone, task sets, and verifier. Results report exact execution success rates and provide the numerical values behind Figure 3.

Dataset	Prior	ReAct-style one-shot	Reflexion-style retry	VGS
App-200				
	SFT	0.740	0.940	0.975
	Vanilla DPO	0.750	0.950	0.980
	mDPO	0.785	0.975	1.000
ProcTHOR				
	SFT	0.730	0.890	0.940
	Vanilla DPO	0.750	0.970	0.985
	mDPO	0.770	1.000	1.000

C.4 VGS Hyperparameter Sensitivity. Table C.5 provides a sensitivity check on the internal cost of VGS on App-200. We vary the search iteration budget and search depth, and report the resulting search-side overhead only, excluding LLM decoding time. Across the tested ranges, the added cost remains at the millisecond level, increasing from 7ms to 42ms when the iteration budget grows from 8 to 160, and from 18ms to 34ms when the search depth grows from 4 to 24. This result supports the main latency analysis: even when the VGS search budget is enlarged, the extra cost introduced by the repair module itself remains small compared with model decoding time.

C.5 Detailed Inference Time Composition. Table C.6 gives a more detailed view of where inference time is spent. In addition to total runtime and LLM decoding time, it separately reports rule-checking time from the symbolic verifier and VGS control time from the search procedure itself.

Table C.5: VGS hyperparameter sensitivity on App-200. The table reports only search-side overhead and excludes LLM decoding time. Even with larger search budgets or deeper search, the added cost remains at the millisecond level.

Hyperparameter Setting	Search-side Overhead
MCTS iterations ($8 \rightarrow 160$, depth = 18)	increases from 7ms to 42ms
MCTS depth ($4 \rightarrow 24$, iter = 96)	increases from 18ms to 34ms

It also shows how often Reflexion-style retry needs an extra decoding round and how often VGS is triggered. Across both App-200 and ProcTHOR, the main pattern is the same: most time is spent on LLM decoding, while the extra control cost introduced by VGS remains small.

Table C.6: Detailed inference time composition on App-200 and ProcTHOR. Time columns are average seconds per task. 'Rule checking' is the time spent on symbolic verifier checks, and 'VGS control' is the time spent on the search procedure itself, excluding LLM decoding. 'Extra decoding rounds' is the average number of additional decoding attempts used by Reflexion-style retry, and 'Repair trigger rate' is the fraction of tasks on which VGS is activated.

Dataset	Method	Final success	Total time (s)	LLM decoding (s)	Rule checking (s)	VGS control (s)	Extra decoding rounds	Repair trigger rate
App-200	ReAct-style one-shot	0.820	11.596	11.591	0.001	0.000	0.000	–
App-200	Reflexion-style retry	0.960	13.405	13.400	0.001	0.000	0.160	–
App-200	VGS	0.980	11.628	11.583	0.026	0.015	0.000	0.180
ProcTHOR	ReAct-style one-shot	0.820	10.287	10.282	0.001	0.000	0.000	–
ProcTHOR	Reflexion-style retry	0.990	12.223	12.217	0.001	0.000	0.180	–
ProcTHOR	VGS	1.000	10.309	10.264	0.026	0.014	0.000	0.180

C.6 Planner Prior Quality Across Training Methods. Table C.7 reports the full numerical results behind Figure 4. We compare Base, SFT, Vanilla DPO, SimPO, ORPO, and mDPO under one-shot decoding on Qwen2.5-7B and Llama-3.1-8B. Across both backbones, mDPO gives the strongest overall prior quality, with higher execution success and logical completion, together with fewer format errors and severe morphology violations.

Table C.7: **Training-stage planner prior quality (RQ2).** Full numerical results behind Figure 4, evaluated under ReAct-style one-shot decoding. mDPO gives the strongest overall prior quality across both Qwen and Llama backbones.

Backbone	Prior Method	Exact/Exec@100 (↑)	Format Error@100 (↓)	Severe Morph. Viol. (↓)	Logical Comp. (↑)
Qwen2.5-7B	Base	0.0433 ± 0.0153	0.3133 ± 0.0153	0.8000 ± 0.0100	0.1000 ± 0.0100
Qwen2.5-7B	SFT [Zeng et al., 2024]	0.3200 ± 0.0200	0.2867 ± 0.1155	0.6500 ± 0.0608	0.3467 ± 0.0115
Qwen2.5-7B	Vanilla DPO [Rafailov et al., 2023]	0.3367 ± 0.0058	0.0300 ± 0.0300	0.4500 ± 0.0200	0.3900 ± 0.0200
Qwen2.5-7B	SimPO	0.2433 ± 0.0208	0.1100 ± 0.0200	0.4767 ± 0.0153	0.2700 ± 0.0100
Qwen2.5-7B	ORPO	0.1233 ± 0.0569	0.8333 ± 0.0924	0.8500 ± 0.0361	0.1633 ± 0.0252
Qwen2.5-7B	**mDPO (Ours)**	$\mathbf{0.3767 \pm 0.0058}$	$\mathbf{0.0000 \pm 0.0000}$	$\mathbf{0.4233 \pm 0.0252}$	$\mathbf{0.4233 \pm 0.0208}$
Llama-3.1-8B	Base	0.0033 ± 0.0058	0.2667 ± 0.0153	0.8667 ± 0.0208	0.0167 ± 0.0153
Llama-3.1-8B	SFT [Zeng et al., 2024]	0.2633 ± 0.0058	0.2100 ± 0.0300	0.6133 ± 0.0058	0.2833 ± 0.0208
Llama-3.1-8B	Vanilla DPO [Rafailov et al., 2023]	0.3267 ± 0.0058	0.1300 ± 0.0100	0.5467 ± 0.0058	0.3433 ± 0.0252
Llama-3.1-8B	SimPO	0.2067 ± 0.0058	0.2233 ± 0.0208	0.5667 ± 0.0115	0.2367 ± 0.0252
Llama-3.1-8B	ORPO	0.1100 ± 0.0000	0.4967 ± 0.0404	0.9367 ± 0.0321	0.1200 ± 0.0100
Llama-3.1-8B	**mDPO (Ours)**	$\mathbf{0.3500 \pm 0.0000}$	$\mathbf{0.1067 \pm 0.0115}$	$\mathbf{0.5067 \pm 0.0115}$	$\mathbf{0.3733 \pm 0.0208}$

We compare all mappings using the same Qwen2.5-7B base model, SFT initialization, 800/100/100 train, validation, and test split, reference model, training settings, severity scores, and retained preference pairs. We compare continuous margins with the bucket margin and severity weighting variants in Table C.8. The continuous mappings are

$$m_{\log}(s) = 1.10\frac{\log(1+\max(s,0))}{\log 71}, \qquad m_{\text{lin}}(s) = 1.10\frac{\min(\max(s,0),15)}{15}.$$

Table C.8: Continuous and bucketed severity mapping comparison on 100 held-out pairs.

Method	Pair accuracy	Mean reward margin	Pearson	Spearman
Vanilla DPO	100%	23.3139	0.0230	0.1247
Continuous log	100%	23.5993	0.0336	0.1348
Continuous capped linear	100%	23.4669	0.0163	0.1243
Bucket margin only	100%	23.5258	0.0172	0.1158
Weight only	100%	24.2455	0.0388	0.1445
Full mDPO	**100%**	**24.6348**	**0.0615**	**0.1861**

D Case Studies and Execution Traces

This section gives a few concrete examples to complement the main numerical results. Table D.9 shows typical failures from the planner's first attempt, one case where VGS successfully fixes the plan, and one case where the system safely returns failure because no valid plan can be found. These examples show what kinds of errors are still hard and how VGS handles them.

Table D.9: Representative case studies. The table shows typical one-shot planner failures, one successful VGS repair case, and one safe abstention case.

Setting	Case Type	Key Signal	What It Shows
Part I: One-Shot Planner Failures			
SFT one-shot	Parse failure	`parse_ok=False`; `json_array_not_found`	The planner fails to produce a valid structured plan. This is a formatting failure rather than an execution failure.
mDPO one-shot	Topology failure	`graph_infeasible`; missing edge `Lobby→5F_Lobby`	The planner proposes a transition that does not exist in the environment graph. Cross-floor topology remains challenging.
mDPO one-shot	Morphology failure	`morphology_constraint`; `coordination_failure`	The remaining difficult cases are mainly physical or coordination failures rather than formatting mistakes.
Part II: VGS Repair and Safe Abstention			
App-200 with mDPO + VGS	Successful repair	One-shot plan invalid; VGS returns `success_dry_run`	VGS removes an invalid branch, backtracks, and finds a feasible alternative plan.
App-200 with mDPO + VGS	Safe abstention	`mode=search`; no valid plan found	When repair fails, the system returns failure instead of executing an invalid plan.

E Hyperparameter Settings

This appendix summarizes the main hyperparameters for training, inference, search, and data generation. Unless otherwise stated, all experiments use mixed precision (`bfloat16`).

E.1 Training Configurations

Table E.10 shows the training hyperparameters for SFT, Vanilla DPO, and mDPO. Vanilla DPO and mDPO are both trained from the SFT checkpoint.

Table E.10: Training hyperparameters for SFT, Vanilla DPO, and mDPO.

Hyperparameter	SFT	Vanilla DPO	**mDPO (Ours)**
Shared Architecture and Optimization Setup			
Base Model	Qwen2.5-7B-Instruct		
Precision	`bfloat16` (Flash Attention 2 enabled)		
LoRA Target Modules	`all` (q, k, v, o, gate, up, down_proj)		
LoRA Rank (r) / Alpha (α) / Dropout	64 / 128 / 0.05		
Global Batch Size	8		
Optimizer	AdamW (`adamw_torch`)		
Learning Rate Scheduler	Cosine with 0.1 warmup ratio		
Weight Decay / Max Grad Norm	0.01 / 1.0		
Cutoff Length	4096		
Phase Specific Alignment Parameters			
Peak Learning Rate	1.0×10^{-5}	5.0×10^{-6}	5.0×10^{-6}
Training Epochs	2.0	3.0	3.0
Preference Loss	–	Sigmoid	Sigmoid
KL Penalty (β)	–	0.1	0.1
mDPO Margin Mode	–	–	`sample`
mDPO Margin Scale	–	–	1.0
mDPO Margin Weight	–	–	0.5

E.2 Inference and VGS Parameters

Table E.11 shows the decoding settings and VGS parameters used in evaluation. Unless otherwise stated, all evaluations use one-shot decoding with temperature 0.0.

Table E.11: Inference hyperparameters for LLM decoding, VGS, and baseline search protocols.

Component	Parameter Setting
LLM Generation Setup	
Temperature	0.0 (one-shot decoding, `do_sample=False`)
Max New Tokens	2048
Repetition Penalty	1.0 (Disabled)
VGS (Ours)	
MCTS Max Iterations	96
MCTS Rollout Depth	18
Exploration Weight (c_{uct})	1.25
Fallback Mechanism	Validity-first hard pruning ($\mathcal{V}_{\text{feasible}} = 0$)
Max Retries (if Search yields empty)	2
Baseline Agent Protocols	
Reflexion (Retry) Max Attempts	2 (App/Graph) / 3 (RWARE)
Tree of Thoughts (ToT) Iterations / Depth	96 / 18
ReAct / Seq2Seq / BUTLERg	One-shot decoding (Max attempts = 0)

E.3 Data Generation and Auto Evaluation

For Stage 1 candidate generation, we use `gemini-2.5-flash` with temperature 0.4. The script keeps up to 4 top-k candidates and uses a minimum soft utility gap of 2.0. For Stage 2 verification, we first apply the symbolic verifier and only use an LLM fallback when rule based checking is not enough. In the released code, this fallback model is set to `claude-3.7-sonnet` with temperature 0.1 by default.

F Full Prompt Inventory and Agent Configurations

This section shows the main prompt templates used in our pipeline. To support reproducibility, we include the prompts used for data generation, runtime planning, symbolic guidance, and evaluation.

F.1 Data Generation and Preference Pair Construction

We use two prompts to build preference pairs (y^+, y^-). The first prompt generates valid reference plans, and the second prompt generates plausible but flawed alternatives.

G.1.1 Main Planning Prompt (For τ_w Generation)

```
You are a planner for heterogeneous embodied agents on a symbolic campus graph.
**Goal**: Produce a valid executable transport plan with minimal cost.
**Robot Profiles**:
- chassis_car (Max 10kg, Elevators: Yes, Stairscatastrophic: No)
- dog_robot (Max 2kg, Elevators: No, Stairs: Yes)
- delivery_drone (Max 5kg, Outdoor Only)
**Action Schema**: [navigate, call_elevator, climb_stairs, pick_up,
put_down, handover]
**Hard Constraints**:
- The FIRST step must be pick_up at <START_NODE>.
- The FINAL step must be put_down at <END_NODE>.
- Item carrying requires sufficient payload capacity.
- Drone must stay on macro outdoor nodes.
**Output Requirements**:
1. Reasoning inside <think>...</think> (max 6 lines).
2. Final JSON plan inside <answer>...</answer>.
```

G.1.2 Rejected Plan Construction Prompt (For τ_l Generation)

```
[BASE_PROMPT]
[Reference Valid Plan]: <CHOSEN_PLAN_JSON>
[Task]: Create a PLAUSIBLE but INVALID alternative plan.
- Intentionally contain error families: <TARGET_ERROR_DETAILS_BULLETS>
- Keep same JSON schema and step style.
- The invalid plan must still be a NON-EMPTY plan.
Output reasoning in <think> and the invalid JSON in <answer>.
```

F.2 Inference Time System Prompts

For the App benchmark, we use separate prompts for a Lead Planner and an Action Executor. The Lead Planner plans the task at a high level, while the Action Executor converts structured actions into tool calls.

G.2.1 Lead Planner System Prompt (App Benchmark)

```
You are the Lead Planner for the Ironlings Heterogeneous Robot System.
### Environment Context:
[Robot_Station_5F], [Elevator_Lobby_1F/2F/4F/5F], [Starbucks_Outdoor],
[Info_Hub_4F] (Designated Handover), [User_Desk_Internal].
### Agent HARD CONSTRAINTS:
- Chassis: 1F/2F/4F/5F via elevator. PROHIBITED from Narrow Corridor (User Desk).
- Dog: Strictly confined to 4F. CANNOT USE ELEVATORS.
Solely responsible for desk.
### Reasoning Instruction:
Inside <think>, perform Spatial Analysis, Handover Logic, and Pathfinding.
```

G.2.2 Action Executor System Prompt

```
You are the Action Executor. Translate structured JSON to Tool Calls.
### Available Tools:
1. chassis_navigate(target); 2. dog_navigate(target);
3. arm_manage(action, target) [grasp, transfer_to_dog, reset];
4. elevator_control(command) [goto_1F, call_to_4F, etc.]; 5. terminate(status).
### Strategy:
Match 'robot' to tool category. If action is 'grasp/transfer',
MUST use arm_manage.
```

F.3 Runtime Guidance and Symbolic Feedback

This module represents the core of inference stage correction, providing the LLM with deterministic feasibility hints and failure-recovery context.

G.3.1 Symbolic Feasibility Guidance (Graph Runtime)

```
### Symbolic Feasibility Hints
- <ROBOT>: feasible exact single-robot route; cost=<COST>, hops=<HOPS>.
- <ROBOT>: <STATUS> (e.g., "infeasible: cannot climb stairs").
- Best exact single-robot option: <BEST_ROBOT>.
- Route-node hint (route_hint mode): <BEST_PATH_NODES>.
- Avoid multi-robot handovers unless single-robot task is impossible.
```

G.3.2 Replanning Context (On VGS Failure)

```
### Replanning Context
- Previous failure: <FAILURE_DETAIL>
- Failed steps: <FAILED_STEPS>
Fix the broken transitions instead of repeating the same route skeleton.
Do not reuse an invalid handover, teleport, or wrong-floor transition.
```

F.4 Evaluation and Audit Prompts

For evaluation and additional logic checking, we also use separate audit prompts. These prompts are only used for checking and do not change the main symbolic execution results.

G.4.1 Logic Integrity Audit Prompt (Ignore Format Errors)

```
You are a strict evaluator. Ignore JSON/XML format errors.
Infer the intended plan logic from raw text.
Judge whether the plan logic can complete the task:
Instruction: <INSTRUCTION>; Start: <START_NODE>; End: <END_NODE>.
Return JSON: {"can_complete_task": "yes|no", "major_errors": [...],
"reason": "..."}
```

G Baseline Details

G.1 Unified Comparison Protocol. All baselines are evaluated under the same executable protocol: the same task IDs, prompt format, symbolic rules, verifier, and success criterion.

G.2 Inference Baselines. We include three representative inference-time baselines:

- **ReAct-style direct planning** [Yao et al., 2023b, Yang et al., 2024]. This baseline generates one complete plan in a single pass from the task instruction. It does not perform search-based repair after generation, so its result reflects the raw quality of first pass planning.

- **Reflexion-style retry** [Shinn et al., 2023, Pan et al., 2024]. When the first plan fails, the model performs another full decoding attempt and regenerates a new plan. This baseline tests whether simple retry can recover failures without explicit symbolic search control.
- **ToT-style search** [Yao et al., 2023a]. Tree-of-Thoughts expands multiple reasoning branches at inference time and then selects a candidate through search. It is a strong search-based baseline, but it does not use our verifier-guided pruning design.

G.3 Planner-Training Baselines (Planner Priors). We also compare training methods that produce different planner priors (planner priors: planners before search/repair):

- **Base LLM**. The original pretrained model without task-specific alignment.
- **SFT (Supervised Fine-Tuning)** [Zeng et al., 2024]. The model is fine-tuned on task demonstrations, providing a standard supervised initialization for downstream preference learning.
- **Vanilla DPO (Direct Preference Optimization)** [Rafailov et al., 2023]. This method aligns the model with pairwise preferences, but treats different failure severities with the same preference margin.
- **SimPO** [Meng et al., 2024]. A preference-learning baseline with a simplified pairwise optimization objective; included to compare with DPO-style training under the same data pipeline.
- **ORPO**. [Hong et al., 2024]. A preference-learning baseline using an odds-ratio-style objective; included to test another alignment formulation under the same evaluation protocol.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and introduction explicitly state our contributions (mDPO and VGS) and scope, which are comprehensively supported by the experimental results in Section 4 and the theoretical bounds in Appendix A.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Appendix C.7 discusses the limitations of symbolic evaluation, the absence of real robot experiments, and the difficulty of transfer to unseen buildings. Appendix C.8 also discusses dependence on the correctness and completeness of the verifier.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

Answer: [Yes]

Justification: Section 3.5 presents the gradient analysis, and Appendix A provides the assumptions and proofs for the preference boundary, gradient bounds, and parameter updates in Propositions A.1–A.3.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: Appendix B describes data construction and VGS, Appendix E reports training and inference settings, and Appendix F provides prompt templates. Case studies and the baseline protocols are provided in Appendices D and G.

Guidelines:

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

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

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

   Answer: [Yes]

   Justification: The MAPS code and App-200 data are publicly available at `https://github.com/AmberGTP5/MAPS`. The paper provides the data construction procedure, hyperparameters, prompts, and evaluation protocols.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Section 4.1 describes the datasets, evaluation metrics, and shared experimental protocol. Appendix E specifies the optimizer, learning rates, batch size, training epochs, decoding settings, and search parameters, while Appendix G describes the baselines.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Table 1 reports the mean and standard deviation over three runs. Appendix C.2 reports exact McNemar tests on matched task outcomes, with $p < 0.001$ for the reported comparisons.

Guidelines:

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Appendix E reports precision and training settings, and Section 4.4 reports inference latency.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: Our research focuses on enhancing the safety and logic of robotic planning algorithms and strictly conforms to the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

Answer: [Yes]

Justification: Appendix C.8 discusses potential benefits for service robots and warehouse systems, possible misuse, dependence on verifier correctness, and the limitations of transferring symbolic evaluation results to physical deployment.

Guidelines:

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

11. **Safeguards**

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

Answer: [N/A]

Justification: Our released datasets and specialized planning models are designed for specific embodied control benchmarks and do not inherently pose a high risk for dual-use or malicious generation.

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: All utilized benchmarks (ALFWorld, ProcTHOR, RWARE) and base language models (Qwen, Llama) are properly cited and used strictly within their respective open-source licenses.

Guidelines:

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

13. **New assets**

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

Answer: [Yes]

Justification: The new App-200 benchmark and the multi-agent synthesized preference datasets are thoroughly documented in the supplementary material.

Guidelines:

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

14. **Crowdsourcing and research with human subjects**

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

Answer: [N/A]

Justification: This work entirely relies on simulated environments and LLM-synthesized data; no human subjects or crowdsourcing were involved.

Guidelines:

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

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

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

Answer: [N/A]

Justification: No human subjects were involved in this research.

Guidelines:

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

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [Yes]

Justification: Appendix E.3 describes the use of `gemini-2.5-flash` for candidate generation and verification fallback. Appendix F provides the prompts used for data construction, planning, and evaluation.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.