TGPO: Trace-Guided Policy Optimization for Robot Task Planning via Verifiable Subgoal Generation

Zhihong Liu[1,3*] **Yang Li**[1,4*] **Renming Huang**[1,2]
Chendong Zeng[1,2] **Cewu Lu**[1,2] **Panpan Cai**[1,2†]
[1]Shanghai Innovation Institute [2]Shanghai Jiao Tong University
[3]Xi'an Jiaotong University [4]Nankai University

Abstract

Robot task planning in real-world environments requires mapping abstract natural language instructions to executable action sequences under long horizons and complex constraints. While large language models (LLMs) provide strong commonsense reasoning, they often fail to generate reliable and feasible plans. In contrast, symbolic planners ensure feasibility and optimality but require well-specified goals and cannot directly interpret high-level human intent. We formulate robot task planning as learning to generate *verifiable subgoals* in the Planning Domain Definition Language (PDDL), bridging language understanding and symbolic planning. To address the challenges of sparse and noisy supervision, we propose Trace-Guided Policy Optimization (TGPO), a reinforcement learning framework that improves structured subgoal generation through (i) verifier-grounded rewards, (ii) external correction of intermediate reasoning traces, and (iii) constrained policy updates that incorporate corrected traces into training. We evaluate TGPO on large-scale household planning tasks with long horizons, abstract instructions, and complex constraints. TGPO significantly outperforms prompting-based and reinforcement learning baselines, with the largest gains on abstract tasks. Furthermore, TGPO integrates naturally with symbolic planners and language-conditioned executors, enabling robust long-horizon planning and combinatorial generalization in realistic environments.[3]

1 Introduction

Long-horizon planning from natural language remains a fundamental challenge for robotic systems. Real-world instructions are often abstract, underspecified, and require reasoning over extended horizons under complex constraints [1, 2]. While large language models (LLMs) [3, 4, 5] exhibit strong commonsense reasoning, they struggle to produce reliable and executable plans [6, 7, 8]. This failure stems from several factors: (i) lack of explicit constraint modeling, leading to infeasible or inconsistent action sequences; (ii) compounding errors over long autoregressive generations, where early mistakes propagate and invalidate downstream steps; and (iii) weak grounding to environment state, especially in large-scale scenes. As a result, planning performance degrades sharply as task complexity and horizon increase. In contrast, symbolic planners [9, 10, 11] provide guarantees on feasibility and optimality, but require well-specified symbolic goals and cannot directly interpret high-level human intent. Bridging language understanding and symbolic reasoning is therefore central to scalable robot planning.

*Equal contribution.
†Corresponding author: cai_panpan@sjtu.edu.cn.
[3]Project page: https://tgpo2026.github.io/TGPO/.

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

A common paradigm for integrating LLMs with symbolic planning [12, 13, 14, 15] is to have LLMs translate a language instruction into a symbolic goal specification, e.g., in Planning Domain Definition Language (PDDL) [16], which is then solved by a classical planner [17]. While effective for simple tasks, directly generating a single holistic goal becomes brittle as task complexity increases. Such goal specifications often fail to capture temporal dependencies among subtasks and omit key task elements implicit in the instruction; solving the planning problem can become prohibitively expensive or even impossible. A natural remedy is to decompose the task into intermediate subgoals [18, 19]. Subgoals make key task elements and their temporal dependencies more explicit, and reduce solver complexity by breaking a long-horizon problem into smaller, modular subproblems. However, decomposition shifts the bottleneck to the *formulation* of subgoals: the model must map an abstract instruction to a sequence of grounded, composable subgoals that are individually feasible, consistently chained without conflict or redundancy, and jointly sufficient to satisfy the implicit task intent.

In this work, we propose to *learn* subgoal formulation as a structured policy. We introduce a foundation model that generates *verifiable subgoals*, which, when coupled with a symbolic planner, enables scalable robot task planning in large-scale environments with long horizons, complex constraints, and abstract instructions. Our key idea is to directly optimize the policy for producing subgoal sequences that are feasible, consistent, and sufficient for task completion, rather than relying on prompting or post-hoc correction. We formulate this as a reinforcement learning problem, where the policy is trained to maximize verifier-grounded rewards reflecting subgoal feasibility and overall task success. However, this learning problem is inherently challenging: reward signals are *sparse*, since small errors in subgoal formulation often invalidate entire plans; *noisy*, as satisfaction of implicit task intent is difficult to reliably evaluate; and *delayed*, making credit assignment over intermediate subgoals particularly difficult.

To address these challenges, we propose **Trace-Guided Policy Optimization (TGPO)**, a reinforcement learning algorithm for structured subgoal generation. TGPO is designed to tackle the key difficulties of sparse, noisy, and delayed feedback in subgoal formulation by introducing structured learning signals at the level of intermediate reasoning. Specifically, TGPO incorporates three components: (i) a *verifier-grounded reward* that provides outcome-level supervision based on subgoal feasibility and task completion; (ii) *trace-guided refinement*, where an auxiliary model identifies and corrects errors in intermediate reasoning traces, providing fine-grained guidance beyond scalar rewards; and (iii) *constrained policy optimization*, which integrates corrected traces into training via token-level constraints while preserving autoregressive generation. Together, these mechanisms transform sparse and delayed rewards into dense, structured supervision, significantly improving exploration efficiency and credit assignment over long-horizon reasoning processes.

We evaluate TGPO on a diverse suite of long-horizon planning tasks spanning varying levels of abstraction, horizon length, scene complexity, and environmental uncertainty. TGPO consistently outperforms state-of-the-art prompting-based planners [20, 13, 12, 14] and existing reinforcement learning methods [21, 22]. It achieves particularly strong gains on complex, long-horizon tasks with abstract instructions and large-scale scenes, where implicit intent inference and compositional reasoning are critical. Moreover, TGPO generalizes effectively to novel tasks and environments, remains robust under stochastic dynamics, and integrates seamlessly with low-level executors, enabling scalable end-to-end robotic systems. Qualitative results are available on the project page.

Our contributions are threefold: *(1) Learning-based subgoal formulation.* We cast long-horizon planning as learning a policy that generates verifiable subgoals, moving beyond prompting-based decomposition. *(2) Trace-guided policy optimization.* We propose TGPO, a reinforcement learning algorithm that leverages external trace correction to improve structured generation under sparse and noisy rewards. *(3) Scalable long-horizon planning.* We demonstrate that TGPO enables robust planning in large-scale environments with long horizons and abstract instructions, significantly outperforming existing methods.

2 Related Work

LLMs for planning. Recent work leverages large language models (LLMs) for task planning by prompting them to generate action sequences or intermediate representations [23, 24, 25, 26]. Approaches such as SayPlan [20] and DELTA [13] decompose long-horizon tasks into subproblems, often using structured inputs like scene graphs [27, 28]. While these methods demonstrate strong

performance in simple settings, they rely on general-purpose LLMs at inference time and typically suffer from limited scalability as task ambiguity and constraint complexity increase.

LLM–symbolic planning integration. Another line of work combines LLMs with symbolic planners [29, 30, 31, 32, 33], typically using LLMs to generate PDDL [9] goals or problem specifications that are subsequently solved by off-the-shelf planners [17] (e.g., LLM+P [12], NL2Plan [14]). These methods benefit from the verifiability and optimality of symbolic planning, but depend on the LLM's ability to produce correct symbolic formulations [34]. As a result, performance is often bottlenecked by the quality of prompting and does not directly improve with experience. In contrast, our approach treats subgoal generation as a learnable problem and optimizes it through reinforcement learning.

Reinforcement learning for LLM. Reinforcement learning has been widely used to align and improve LLMs via methods such as RLHF [35], GRPO [21], and Reinforce++ [22]. These approaches typically rely on outcome-based reward signals and optimize generation at the token level. Consequently, they face significant challenges in abstract tasks characterized by sparse, ambiguous, and noisy rewards [36, 37]. In particular, credit assignment over intermediate reasoning steps remains difficult. Our method addresses this by incorporating external feedback through trace-level correction.

Process supervision and external guidance. Recent work explores supervising intermediate reasoning processes, such as chain-of-thought supervision [38, 37, 39], self-refinement [40, 41], and reflection-based [42, 43] methods. These approaches improve reasoning quality by modifying intermediate outputs, but are usually applied at inference time or through supervised learning. In contrast, we integrate trace correction directly into reinforcement learning, enabling the policy to learn from corrected reasoning trajectories during optimization.

Summary. In contrast to prior work that relies on prompting or post-hoc refinement, we formulate subgoal generation as a learnable policy (Section 3) and introduce a trace-guided reinforcement learning framework (Section 4) for optimizing structured reasoning under verifiable constraints.

3 Problem Formulation

We consider long-horizon planning from natural language under structured constraints. Given a potentially abstract language instruction q, a structured representation of a large environment $\mathcal{S}$, e.g., a scene graph [28, 44], and a PDDL domain $\mathcal{D}$ that symbolically specifies the robot's capabilities, the goal is to produce an action plan $A = \{a_k\}_{k=1}^{K}$ that satisfies the user's intent while remaining feasible under $\mathcal{D}$.

Structured subgoal generation. Instead of directly generating action sequences or a single PDDL goal, we formulate planning as generating a sequence of intermediate PDDL subgoals:

$$(T, G) = \pi_\theta(q, \mathcal{S}), \tag{1}$$

where $T = \{t_k\}_{k=1}^{K}$ is a sequence of natural-language decomposition steps, and $G = \{g_k\}_{k=1}^{K}$ is a sequence of grounded symbolic subgoals. Each subgoal g_k is expressed in a structured form (i.e., PDDL) and is associated with a subset of relevant objects.

Solver-grounded execution. Each subgoal g_k defines a subproblem that can be solved by a symbolic planner:

$$a_k = \textsc{Solve}(\mathcal{D}, \mathcal{S}_k, g_k), \tag{2}$$

where $\mathcal{D}$ denotes the planning domain and $\mathcal{S}_k$ is the updated environment state after executing previous subgoals. The final plan $A = \{a_k\}_{k=1}^{K}$ is obtained by composing the subplans.

Planning objectives. We define a reward function based on two criteria: (i) *feasibility*, which checks whether each subgoal is solvable by the planner, and (ii) *task completion*, which evaluates whether the composed plan satisfies the instruction. Formally, the reward for an output (T, G) is:

$$R(T, G) = R_{\text{feas}}(G) \cdot R_{\text{comp}}(T, G), \tag{3}$$

where $R_{\text{feas}} \in \{0, 1\}$ indicates whether all subgoals are solvable, and $R_{\text{comp}} \in [0, 1]$ measures task completion.

The learning challenge. Optimizing π_θ under this objective is challenging for two reasons. First, the reward is sparse and noisy, especially for abstract tasks where goal completion is difficult to verify.

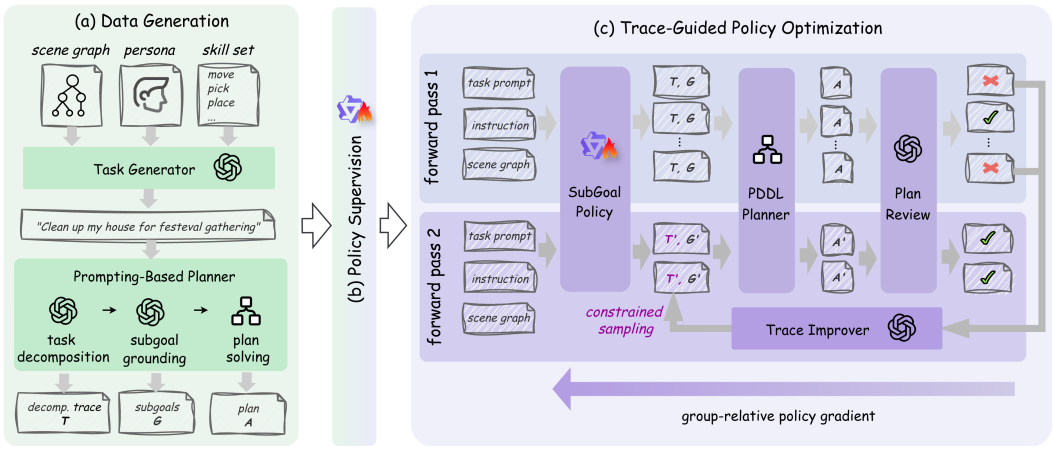


Figure 1: Overview of TGPO. *(a) Data Generation*: Construction of a long-horizon planning dataset with task decomposition traces and grounded subgoals. *(b) Policy Initialization*: Supervised fine-tuning (SFT) to initialize a structured generation policy. *(c) Trace-Guided Policy Optimization*: A reinforcement learning framework that improves structured generation by incorporating external trace correction and constrained sampling, enabling effective learning under sparse and noisy rewards.

Second, errors often arise from intermediate reasoning steps (e.g., missing, overlapping or misordered subgoals), making credit assignment over the sequence difficult. These challenges motivate the trace-guided optimization strategy introduced in Section 4.

4 Trace-Guided Policy Optimization

We propose **Trace-Guided Policy Optimization (TGPO)**, a reinforcement learning algorithm for structured subgoal generation under sparse and noisy rewards. TGPO builds upon group-relative policy optimization (GRPO) [21], but differs in how learning signals are constructed and utilized: it introduces a two-pass rollout with external trace correction, and performs policy optimization over both free and trace-constrained trajectories, enabling structured credit assignment over intermediate reasoning steps.

Let π_θ denote the policy that generates structured outputs (T, G) given instruction q and environment $\mathcal{S}$. Following GRPO, we sample a group of N candidate outputs $\{o_n\}_{n=1}^N$ and compute group-relative advantages based on their rewards $R(o_n)$. The standard GRPO objective is:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}\left[\frac{1}{N}\sum_{n=1}^{N}\min\left(\rho_n A_n, \text{clip}(\rho_n, 1-\epsilon, 1+\epsilon)A_n\right)\right], \tag{4}$$

where $\rho_n = \frac{\pi_\theta(o_n)}{\pi_{\theta_{\text{old}}}(o_n)}$ and A_n is the group-relative advantage. While effective, this formulation treats each output as a whole and relies solely on scalar rewards, making it difficult to correct errors arising from intermediate reasoning steps.

TGPO augments GRPO with a second rollout pass that explicitly corrects intermediate reasoning traces (Fig. 1(c)).

Free rollouts. We sample a group of N trajectories from the current policy π_θ:

$$o_n = (T_n, G_n) \sim \pi_\theta(\cdot \mid q, \mathcal{S}), \quad n = 1, \ldots, N, \tag{5}$$

where each output o_n consists of a decomposition trace $T_n = \{t_{n,k}\}_{k=1}^{K_n}$ and a sequence of grounded subgoals $G_n = \{g_{n,k}\}_{k=1}^{K_n}$. The policy π_θ defines an autoregressive distribution over tokens that jointly generate (T_n, G_n). Each trajectory is evaluated using the verifier-grounded reward $R(o_n)$.

Trace correction. We identify a subset of low-quality trajectories based on their rewards:

$$\mathcal{I}_{\text{fail}} = \{n \mid R(o_n) < \tau\}, \tag{6}$$

where τ is a predefined threshold. For each $n \in \mathcal{I}_{\text{fail}}$, we apply a trace correction operator $\mathcal{C}$ that revises the decomposition trace:

$$\tilde{T}_n = \mathcal{C}(T_n; q, \mathcal{S}), \tag{7}$$

where $\mathcal{C}$ is implemented by an auxiliary model that conditions on the instruction and environment, details are provided in Appendix B. The corrected trace $\tilde{T}_n = \{\tilde{t}_{n,k}\}_{k=1}^{\tilde{K}_n}$ is intended to address common failure modes in structured reasoning, including missing subtasks, incorrect temporal ordering, and incomplete grounding of task intent. Importantly, this step modifies only the intermediate reasoning trace, while leaving subgoal generation to the policy in the subsequent rollout.

Constrained rollouts. For each corrected trace $\tilde{T}_n$, we perform a second rollout by conditioning the policy on $\tilde{T}_n$ and generating the corresponding subgoals:

$$\tilde{o}_n = (\tilde{T}_n, \tilde{G}_n) \sim \pi_\theta(\cdot \mid q, \mathcal{S}; \tilde{T}_n). \tag{8}$$

Concretely, we implement this as token-level constrained sampling: tokens corresponding to the trace $\tilde{T}_n$ are *forced* during decoding, while the remaining tokens (i.e., subgoals) are generated autoregressively. The resulting trajectories $\{\tilde{o}_n\}$ are then evaluated using the reward $R(\tilde{o}_n)$.

Policy optimization. We combine trajectories from both rollout passes into an augmented group:

$$\mathcal{G} = \{o_n\}_{n=1}^{N} \cup \{\tilde{o}_n\}_{n \in \mathcal{I}_{\text{fail}}}, \tag{9}$$

where $\{o_n\}$ are trajectories from the free rollout and $\{\tilde{o}_n\}$ are generated via constrained rollout with corrected traces. We compute group-relative advantages over $\mathcal{G}$ and apply a similar clipped objective as in Eq. (4):

$$\mathcal{L}_{\text{TGPO}}(\theta) = \mathbb{E}\left[\frac{1}{|\mathcal{G}|}\sum_{o \in \mathcal{G}} \min\left(\rho(o)A(o), \text{clip}(\rho(o), 1-\epsilon, 1+\epsilon)A(o)\right)\right], \tag{10}$$

where $\rho(o) = \frac{\pi_\theta(o)}{\pi_{\theta_{\text{old}}}(o)}$.

Note that, during constrained rollout, TGPO computes the log-probability of the full sequence, so that policy gradients are computed with respect to all tokens in the sequence, including the constrained trace. Moreover, policy gradients in TGPO are computed over both original and trace-corrected trajectories. This effectively incorporates external corrections into the optimization process. Note that, while constrained sampling introduces a deviation from the policy's native exploration distribution, we do not explicitly correct for this mismatch via importance weighting. Instead, the bias is naturally mitigated in practice because constraints are applied only to the subset of failed trajectories and are restricted to intermediate reasoning tokens, leaving the remaining generation process on-policy.

5 Application to Large-Scale Household Planning

We instantiate the proposed framework for long-horizon household planning by grounding the general formulation in Section 3 in mobile manipulation tasks in large indoor environments. Each task is specified by a natural language instruction q and a structured environment scene graph $\mathcal{S} = (V, E, X)$. Here, V denotes entities such as rooms, furniture, and objects; E encodes spatial and semantic relations (e.g., `in`, `on`); and X specifies symbolic state attributes (e.g., open/closed, clean/dirty). Further details on the scene graph representation are provided in Appendix A.1. Planning is performed over a household PDDL domain $\mathcal{D}$ that defines a wide set of predicates and operator schemas for mobile manipulation, details are provided in Appendix J. Given $(q, \mathcal{S})$, the objective is to generate a sequence of grounded subgoals that are valid under $\mathcal{D}$ and can be composed into an executable mobile manipulation plan via a symbolic planner.

5.1 Training Data Generation

We construct a large-scale dataset by jointly sampling household environments and tasks with controlled diversity. Fig. 1(a) visualizes the data generation pipeline.

(a) Scene distribution. We construct the scene distribution from the Habitat Synthetic Scene Dataset (HSSD) [45], which contains high-quality 3D indoor environments with realistic layouts and a diverse

set of real-world objects. We directly reuse the textual scene graphs provided by HSSD. To increase diversity while preserving realism, we further augment these base scene graphs by injecting additional objects and attributes conditioned on task requirements. This augmentation maintains plausible spatial and semantic consistency, resulting in a large set of scene graphs that reflect realistic environments while supporting diverse planning scenarios.

(b) Task distribution. As illustrated in Fig. 1(a), given a base scene $\mathcal{S}$ and a skill set derived from the planning domain $\mathcal{D}$, we synthesize task instructions using a large language model jointly conditioned on $\mathcal{S}$, $\mathcal{D}$, and a sampled user *persona*. Each persona specifies attributes such as occupation, preferences, and situational context, inducing diverse task intents grounded in the same environment. This persona-conditioned generation enables controlled diversity in task distribution, producing instructions that vary not only in surface form but also in underlying intent and abstraction. In particular, we explicitly vary the level of abstraction, ranging from explicit commands to high-level intent descriptions that require implicit reasoning and grounding. Further details regarding the synthetic dataset are provided in Appendix H.

(c) Annotation and filtering. For each $(q, \mathcal{S})$, we generate structured outputs (T, G)—including decomposition traces and grounded subgoals—using a prompting-based planner. This planner employs a general-purpose LLM (e.g., GPT [3]) to produce outputs in the same format as the TGPO policy (detailed in Appendix A), but without task-specific training. We retain only samples that satisfy both feasibility (all subgoals are solvable) and completion (the resulting plan fulfills the task intent), yielding high-quality supervision for learning.

5.2 Training Pipeline

We adopt a two-stage training pipeline consisting of supervised fine-tuning (SFT, Fig. 1(b)) followed by reinforcement learning with TGPO (Fig. 1(c)). In the first stage, we initialize the policy via supervised fine-tuning (SFT) on the constructed dataset using Qwen2.5-7B [46] as the base model, enabling the policy to learn fundamental capabilities for task decomposition and subgoal grounding. In the second stage, we apply TGPO to further optimize the policy via trace-guided reinforcement, refining intermediate reasoning, increasing task completion, and enforcing subgoal feasibility. Further details regarding the training are provided in Appendix B.

6 Experiments

Benchmarks. We evaluate our method on four complementary benchmarks.

(a) In-distribution synthetic tasks. We evaluate on novel in-distribution (ID) tasks drawn from the same scene and task generation processes as the training data. Importantly, while the training set includes only tasks that are solvable by a prompting-based planner, the test set is not subject to this filtering, and therefore is a broader and potentially more challenging set. The test tasks are partitioned into three categories:

- **Easy** (146 tasks / 109 scenes): instructions explicitly specify the required *action* and the *objects of interest*, leading to short-horizon plans with few constraints. Example: "I'm in the bathroom, can you bring me a towel?".
- **Complex** (94 tasks / 75 scenes): explicit instructions with long-horizon (15.6 steps on average), constraint-heavy planning, typically involving multiple entities (*e.g.*, objects, devices, and furniture) and cross-area coordination. Example: "Retrieve the portable induction stove and hot-pot saucepan from the kitchen cabinet, place them on the kitchen island, fill the saucepan with water, and turn on the stove.".
- **Abstract** (108 tasks / 45 scenes): instructions that express user needs at an abstract level (6.4 words on average) without explicitly specifying the target objects or intermediate subgoals, requiring the model to infer complex and implicit intent; the planning horizon is comparable to the Complex subset (20 steps on average). Example: "Clean up the house for a festival gathering.".

(b) Human tasks. We evaluate on human-authored instructions collected on scenes generated from the same pipeline. Participants are presented with visualized scene graphs and asked to propose tasks. After manual filtering for quality, the resulting test set consists of 50 tasks. Human instructions are

highly concise—averaging only 5.7 words—comprising a mix of explicit short-horizon tasks and abstract long-horizon tasks. See details in Appendix F.3.

(c) PARTNR [47]. We evaluate on the PARTNR benchmark, a public household task planning benchmark. Despite similar planning horizons (15.8 steps on average), PARTNR tasks are explicitly instructed. Scenes are also significantly smaller (65 nodes on average vs. 150 in our setting), including only task-relevant objects.

(d) BEHAVIOR-1K [1]. We further evaluate on BEHAVIOR-1K, a public benchmark for household task planning and skill learning. We use the 50 household tasks from the 2025 BEHAVIOR Challenge. These tasks feature long planning horizons (24.7 steps on average), but with explicit instructions (40.6 words on average) and substantially smaller scene graphs (11.5 nodes on average).

Evaluation protocol and metrics. In Section 6.1-6.3, we evaluate high-level planning independently. In Section 6.4, we further provide a system-level evaluation with low-level execution.

Given a policy output (T, G), we evaluate the following two aspects of the high-level plan:

(a) Feasibility. We assess whether subgoals are executable under the given environment. For synthetic, human, and BEHAVIOR-1K tasks, feasibility is verified using PDDL solvers [17], which check whether all subgoals are solvable. For PARTNR, we instead rely on the simulator to determine feasibility. This defines a feasibility score $S_{\text{feas}}(o_i) \in \{0, 1\}$. $S_{\text{feas}}(o_i) = 1$ means the entire plan is feasible.

(b) Task completion. We evaluate whether the composed plan satisfies the instruction. For synthetic and human tasks, we use an *LLM-as-judge* protocol, LLM-Rubric [48]. We query a large language model multiple times from different evaluation perspectives, including consistency between the final state and the instruction, alignment of decomposed subtasks with the instruction, action-sequence validity, and symbolic state-transition consistency. We then aggregate the scores via a weighted average. Details are provided in Appendix C. For PARTNR and BEHAVIOR-1K, task completion is determined using their simulator's explicit goal, details are provided in Appendix F. This defines a task completion score $S_{\text{comp}}(o_i) \in [0, 1]$.

Combining the above derives the *Success Rate*, written as $\text{SR}(o_i) = S_{\text{feas}}(o_i) \cdot S_{\text{comp}}(o_i)$. We also report the planning time T when applicable.

Baselines. We compare TGPO with three categories of baselines:

(a) Prompting-based planners. SayPlan [20] performs task decomposition and action grounding in natural language, and iteratively re-plans via reflection. DELTA [13] formulates a holistic PDDL problem, and decomposes it into subproblems for efficient solving. LLM+P and NL2Plan generate a PDDL problem and directly solve it. We also include vanilla Gemini-3-Pro and GPT-5 as baselines[4], prompting them to directly generate action plans assisted by generic chain-of-thought (CoT) [38].

(b) Learning-based planners. SFT [35] learns subgoal generation using supervised learning. GRPO [21] applies group-relative policy optimization to learn subgoal generation via RL. Reinforce++ [22] is an RL method based on REINFORCE, with reduced variance. TGPO (E2E) is an ablated version of our method, learning to directly output action sequences. All methods are trained with the same data and optimization settings for fair comparison.

6.1 Performance Comparison with Prompting-Based Planners

Table 1 summarizes performance across all benchmarks. TGPO consistently outperforms prompting-based methods by a large margin, with gains that systematically increase as tasks become more long-horizon, abstract, and ambiguous. This trend highlights the advantage of our learning-based approach over prompting-based methods, particularly in handling implicit intent and maintaining feasibility over long-horizon tasks.

(a) In-distribution tasks. TGPO significantly outperforms prompting-based methods across all in-distribution settings, with improvements over the strongest baseline increasing from 67% on easy tasks to 131% on complex tasks and 213% on abstract tasks, indicating that the advantage of TGPO becomes more pronounced as task complexity and ambiguity increase.

[4]We use `Gemini-3-Pro` (low reasoning setting) and `GPT-5` (no extra reasoning effort).

Table 1: Comparison with prompting-based baselines. Numbers are success rates (%). T denotes the average planning time per task.

		In-Distribution			Out-of-Distribution		
Method	**T (s)**	Easy	Complex	Abstract	Human	PARTNR	BEHAVIOR-1K
GPT-5	8.0	38.3 ± 3.4	24.6 ± 1.0	9.1 ± 3.2	21.7 ± 2.7	69.2 ± 1.2	35.1 ± 1.0
Gemini-3.0-pro	23.5	30.0 ± 2.3	16.8 ± 1.4	19.9 ± 2.9	13.0 ± 3.0	71.8 ± 1.4	26.0 ± 1.1
SayPlan	63.7	47.5 ± 2.8	33.5 ± 2.2	22.4 ± 2.1	26.5 ± 3.4	72.2 ± 1.8	43.8 ± 0.7
DELTA	37.8	52.7 ± 2.6	22.0 ± 2.8	3.9 ± 1.3	26.5 ± 2.1	69.3 ± 1.1	39.3 ± 1.5
LLM+P	12.3	12.6 ± 1.0	2.1 ± 0.4	4.4 ± 0.8	3.9 ± 1.8	67.2 ± 1.7	22.5 ± 1.2
NL2Plan	18.9	12.8 ± 0.9	10.6 ± 0.7	1.7 ± 0.6	4.5 ± 2.0	66.8 ± 1.0	21.0 ± 1.2
TGPO	3.1	**88.1** ± 0.4	**77.3** ± 0.4	**70.2** ± 1.8	**73.6** ± 1.9	**82.6** ± 0.5	**70.3** ± 0.8

Table 2: Comparison with learning-based baselines. Numbers are success rates (%).

	In-Distribution			Out-of-Distribution		
Method	Easy	Complex	Abstract	Human	PARTNR	BEHAVIOR-1K
Qwen2.5-7B (Base)	0.0	0.0	0.0	0.0	27.8 ± 1.0	12.5 ± 0.7
SFT	83.2 ± 0.5	71.1 ± 0.6	38.1 ± 1.7	55.8 ± 2.2	71.7 ± 1.3	63.7 ± 1.1
GRPO	84.1 ± 0.8	76.0 ± 0.2	51.1 ± 2.4	65.9 ± 2.6	76.4 ± 0.9	67.1 ± 0.8
Reinforce++	83.7 ± 0.6	77.1 ± 0.3	54.0 ± 1.5	65.5 ± 3.3	75.2 ± 1.2	68.3 ± 1.3
TGPO (E2E)	72.4 ± 0.5	55.4 ± 0.2	44.1 ± 2.0	54.1 ± 2.2	62.3 ± 0.6	54.4 ± 1.0
TGPO (Subgoal, ours)	**88.1** ± 0.4	**77.3** ± 0.4	**70.2** ± 1.8	**73.6** ± 1.9	**82.6** ± 0.5	**70.3** ± 0.8

(b) Human tasks. The performance of TGPO on human tasks is consistent with that on in-distribution complex and abstract tasks, achieving a 178% improvement over the strongest prompting-based planner, demonstrating robust generalization to real-world task distributions.

(c) PARTNR. On PARTNR, where both tasks and environments are out-of-distribution, TGPO achieves a success rate of 82.6%, demonstrating strong generalization. Since PARTNR tasks are explicitly specified and involve simple constraints and small scene graphs, they are easier for all methods.

(d) BEHAVIOR-1K. TGPO also successfully generalizes to BEHAVIOR-1K, where both environments and tasks differ significantly from the training distribution, achieving a success rate of 70.3%. Compared to the strongest baseline, this corresponds to a 61% improvement. The substantial performance gain highlights the robustness of TGPO under severe distribution shift.

6.2 Performance Comparison with Learning-Based Planners

Table 2 compares TGPO with learning-based baselines, also serving as an ablation study of different training strategies. We also provide a scalability test of TGPO in Appendix D.

(a) Base model. The Qwen2.5-7B base model completely fails on synthetic and human tasks, and achieves marginal success on PARTNR and BEHAVIOR-1K, due to their simplified scene graphs. These results reaffirm that mid-scale open models cannot solve large-scale planning tasks without domain-specific adaptation.

(b) TGPO vs. SFT. Compared to supervised fine-tuning (SFT), TGPO achieves substantially higher performance, especially on complex and abstract tasks. This suggests that reinforcement learning provides a significantly higher performance upper bound by optimizing directly for plan feasibility under environment constraints and task completion under abstract instructions.

(c) TGPO vs. GRPO / Reinforce++. The key difference between TGPO and standard RL methods (GRPO, Reinforce++) lies in the use of trace revision as structured guidance during training. Without trace revision, models perform comparably on simple tasks but degrade rapidly as task complexity and abstraction increase. In contrast, TGPO remains robust. This behavior can be explained by the difficulty of credit assignment under sparse and noisy rewards. For long-horizon tasks, successful trajectories are rare, making exploration inefficient. Trace revision helps the policy quickly discover high-quality trajectories. For abstract tasks, reward signals become ambiguous and noisy; trace revision provides additional reinforcement signals that stabilize policy learning.

(d) TGPO (Subgoal) vs. TGPO (E2E). Comparing TGPO (Subgoal) with TGPO (E2E) highlights the importance of structured outputs and symbolic planning. The end-to-end variant must simultaneously interpret the task, decompose it, and generate a feasible action sequence, which significantly increases the learning burden. Failures often arise from violating constraints in the action sequence, leading to infeasible plans. In contrast, TGPO (Subgoal) separates problem formulation from execution: the policy generates structured subgoals, while a symbolic planner tackles constraint satisfaction. This decomposition enables efficient learning and substantially improves performance on complex tasks.

6.3 Performance under Stochastic Environments and Execution Failures

Real-world robotic systems often operate under stochastic dynamics and execution uncertainty. Under such circumstances, the TGPO policy can be coupled with probabilistic PDDL formulations [49]. In fact, the generated subgoals can be applied to any domain sharing the same predicate and operator definitions. This allows replacing the deterministic domain with a probabilistic PDDL model at inference time, enabling closed-loop planning under stochastic action outcomes. We evaluate this setting by integrating TGPO with online Monte Carlo Tree Search (MCTS) [50] in a stochastic environment with a 10% per-action failure probability. Despite the presence of execution uncertainty, the system achieves 80.5% success on easy tasks, 70.8% on complex tasks, and 62.3% on abstract tasks, validating its effectiveness under stochastic dynamics.

6.4 System-Level Performance of Integrated Planning and Execution

We further evaluate the TGPO policy in a full robotic pipeline by integrating it with low-level executors. We conduct experiments in the BEHAVIOR-1K 3D simulator using its household environments. We construct a set of 10 *recombined tasks* by reordering and recombining previously demonstrated low-level skills into novel task sequences (see Appendix E). No demonstrations are provided for these tasks. This setup shifts the evaluation from imitating demonstrations (as in the BEHAVIOR-1K challenge) to planning and compositional generalization.

The robotic system is provided with textual scene graphs, online visual observations, and task instructions. The TGPO policy acts as a high-level planner that generates a sequence of high-level actions. Each action is executed using either parameterized skills (e.g., navigation) or a neural policy trained via imitation learning for complex primitives (e.g., pouring, opening). The neural policy [51] was trained on long-horizon demonstrations of the original tasks in BEHAVIOR-1K. In our system, however, it is restricted to local execution of primitive skills.

Results show that the holistic neural policy generalizes poorly to recombined tasks, achieving only 23% end-to-end task completion due to its reliance on demonstrated action plans. In contrast, our two-level system preserves strong high-level planning capability, with 90% of generated plans passing feasibility verification, and achieves 36% end-to-end task completion on the recombined tasks. The substantial gap between high-level planning performance and end-to-end execution mainly reflects imperfections in the low-level VLA executor. These results demonstrate that the TGPO policy enables compositional generalization by recombining a fixed set of low-level skills to solve novel long-horizon tasks, providing a principled approach to scaling complex robot behaviors beyond imitating demonstrations. Execution demos and qualitative analysis of the recombined tasks are provided on the project website.

7 Conclusion & Limitations

We present TGPO, a framework for long-horizon robot task planning that learns to generate verifiable subgoals from abstract natural language instructions. By combining a learned subgoal-generation policy with symbolic planners, TGPO enables scalable planning in large environments with complex constraints. TGPO directly optimizes subgoal feasibility, consistency, and task completion, while a trace-guided training scheme improves learning under sparse and noisy supervision. Experiments across diverse benchmarks demonstrate that TGPO significantly outperforms prior prompting-based planners and learning-based methods, with particularly strong gains on complex, long-horizon, and abstract tasks.

Despite these results, several limitations remain. First, TGPO relies on structured scene representations, which may lose information compared to raw visual observations; extending the

framework to operate directly on visual inputs is an important direction. Second, the method assumes a predefined, comprehensive symbolic domain; learning or adapting domains across tasks and environments is a key direction for future work. Third, our current formulation assumes full observability, leaving open the integration of perception uncertainty and active exploration in real-world settings.

Acknowledgement

This work is supported by New Generation Artificial Intelligence-National Science and Technology Major Project (No. 2025ZD0122901).

References

[1] Chengshu Li, Ruohan Zhang, Josiah Wong, Cem Gokmen, Sanjana Srivastava, Roberto Martín-Martín, Chen Wang, Gabrael Levine, Michael Lingelbach, Jiankai Sun, et al. Behavior-1k: A benchmark for embodied ai with 1,000 everyday activities and realistic simulation. In *Conference on Robot Learning*, pages 80–93. PMLR, 2023.

[2] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020.

[3] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024.

[4] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[6] Rui Yang, Hanyang Chen, Junyu Zhang, Mark Zhao, Cheng Qian, Kangrui Wang, Qineng Wang, Teja Venkat Koripella, Marziyeh Movahedi, Manling Li, et al. Embodiedbench: Comprehensive benchmarking multi-modal large language models for vision-driven embodied agents. *arXiv preprint arXiv:2502.09560*, 2025.

[7] Manling Li, Shiyu Zhao, Qineng Wang, Kangrui Wang, Yu Zhou, Sanjana Srivastava, Cem Gokmen, Tony Lee, Li E Li, Ruohan Zhang, et al. Embodied agent interface: Benchmarking llms for embodied decision making. *Advances in Neural Information Processing Systems*, 37:100428–100534, 2024.

[8] Cassie Huang and Li Zhang. On the limit of language models as planning formalizers. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4880–4904, 2025.

[9] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. Pddl—the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

[10] Kutluhan Erol, James A Hendler, and Dana S Nau. Umcp: A sound and complete procedure for hierarchical task-network planning. In *Aips*, volume 94, pages 249–254, 1994.

[11] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[12] 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.

[13] Yuchen Liu, Luigi Palmieri, Sebastian Koch, Ilche Georgievski, and Marco Aiello. Delta: Decomposed efficient long-term robot task planning using large language models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10995–11001. IEEE, 2025.

[14] Elliot Gestrin, Marco Kuhlmann, and Jendrik Seipp. Nl2plan: Robust llm-driven planning from minimal text descriptions. *arXiv preprint arXiv:2405.04215*, 2024.

[15] Gautier Dagan, Frank Keller, and Alex Lascarides. Dynamic planning with a llm. *arXiv preprint arXiv:2308.06391*, 2023.

[16] Constructions Aeronautiques, Adele Howe, Craig Knoblock, ISI Drew McDermott, Ashwin Ram, Manuela Veloso, Daniel Weld, David Wilkins Sri, Anthony Barrett, Dave Christianson, et al. Pddl—the planning domain definition language. *Technical Report, Tech. Rep.*, 1998.

[17] Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[18] Minseo Kwon, Yaesol Kim, and Young J Kim. Fast and accurate task planning using neuro-symbolic language models and multi-level goal decomposition. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16195–16201. IEEE, 2025.

[19] David Bai, Ishika Singh, David Traum, and Jesse Thomason. Twostep: Multi-agent task planning using classical planners and large language models. *arXiv preprint arXiv:2403.17246*, 2024.

[20] Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. In *7th Annual Conference on Robot Learning*.

[21] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[22] Jian Hu, Jason Klein Liu, Haotian Xu, and Wei Shen. Reinforce++: Stabilizing critic-free policy optimization with global advantage normalization, 2025. *URL https://arxiv. org/abs/2501.03262*.

[23] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

[24] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR, 2022.

[25] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International conference on robotics and automation (ICRA)*, pages 9493–9500. IEEE, 2023.

[26] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. *arXiv preprint arXiv:2209.11302*, 2022.

[27] Iro Armeni, Zhi-Yang He, JunYoung Gwak, Amir R Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. 3d scene graph: A structure for unified semantics, 3d space, and camera. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5664–5673, 2019.

[28] Qiao Gu, Ali Kuwajerwala, Sacha Morin, Krishna Murthy Jatavallabhula, Bipasha Sen, Aditya Agarwal, Corban Rivera, William Paul, Kirsty Ellis, Rama Chellappa, et al. Conceptgraphs: Open-vocabulary 3d scene graphs for perception and planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5021–5028. IEEE, 2024.

[29] Mengkang Hu, Yao Mu, Xinmiao Yu, Mingyu Ding, Shiguang Wu, Wenqi Shao, Qiguang Chen, Bin Wang, Yu Qiao, and Ping Luo. Tree-planner: Efficient close-loop task planning with large language models. *arXiv preprint arXiv:2310.08582*, 2023.

[30] Zirui Zhao, Wee Sun Lee, and David Hsu. Large language models as commonsense knowledge for large-scale task planning. *Advances in neural information processing systems*, 36:31967–31987, 2023.

[31] Jason Xinyu Liu, Ziyi Yang, Benjamin Schornstein, Sam Liang, Ifrah Idrees, Stefanie Tellex, and Ankit Shah. Lang2ltl: Translating natural language commands to temporal specification with large language models. In *Workshop on Language and Robotics at CoRL 2022*, 2022.

[32] Lin Guan, Karthik Valmeekam, Sarath Sreedharan, and Subbarao Kambhampati. Leveraging pre-trained large language models to construct and utilize world models for model-based task planning. *Advances in Neural Information Processing Systems*, 36:79081–79094, 2023.

[33] Zhehua Zhou, Jiayang Song, Kunpeng Yao, Zhan Shu, and Lei Ma. Isr-llm: Iterative self-refined large language model for long-horizon sequential task planning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2081–2088. IEEE, 2024.

[34] 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. *Advances in Neural Information Processing Systems*, 36:38975–38987, 2023.

[35] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

[36] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[37] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The twelfth international conference on learning representations*, 2023.

[38] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[39] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[40] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[41] Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1051–1068, Singapore, December 2023. Association for Computational Linguistics.

[42] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[43] Matthew Renze and Erhan Guven. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*, 2024.

[44] Iro Armeni, Zhi-Yang He, JunYoung Gwak, Amir R Zamir, Martin Fischer, Jitendra Malik, and Silvio Savarese. 3d scene graph: A structure for unified semantics, 3d space, and camera. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5664–5673, 2019.

[45] Mukul Khanna, Yongsen Mao, Hanxiao Jiang, Sanjay Haresh, Brennan Shacklett, Dhruv Batra, Alexander Clegg, Eric Undersander, Angel X Chang, and Manolis Savva. Habitat synthetic scenes dataset (hssd-200): An analysis of 3d scene scale and realism tradeoffs for objectgoal navigation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16384–16393, 2024.

[46] Qwen Team. Qwen2.5: A party of foundation models, September 2024.

[47] Matthew Chang, Gunjan Chhablani, Alexander Clegg, Mikael Dallaire Cote, Ruta Desai, Michal Hlavac, Vladimir Karashchuk, Jacob Krantz, Roozbeh Mottaghi, Priyam Parashar, et al. Partnr: A benchmark for planning and reasoning in embodied multi-agent tasks. *arXiv preprint arXiv:2411.00081*, 2024.

[48] Helia Hashemi, Jason Eisner, Corby Rosset, Benjamin Van Durme, and Chris Kedzie. Llm-rubric: A multidimensional, calibrated approach to automated evaluation of natural language texts. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13806–13834, 2024.

[49] Håkan LS Younes and Michael L Littman. Ppddl1. 0: An extension to pddl for expressing planning domains with probabilistic effects. *Techn. Rep. CMU-CS-04-162*, 2(99):12, 2004.

[50] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[51] Ilia Larchenko, Gleb Zarin, and Akash Karnatak. Task adaptation of vision-language-action model: 1st place solution for the 2025 behavior challenge. *arXiv preprint arXiv:2512.06951*, 2025.

A Planning Pipeline Details

This appendix provides comprehensive implementation details of the planning pipeline, which is illustrated in Figure A.1. The pipeline operates as follows: taking a user instruction and an environment scene graph (detailed in Appendix A.1) as inputs, the subgoal policy systematically breaks down the instruction by first generating task decomposition steps (Appendix A.2), followed by a corresponding sequence of grounded subgoals (Appendix A.3). Subsequently, the pipeline processes these subgoals incrementally: for each subgoal, a rule-based Construct$(\cdot)$ procedure (Appendix A.4) translates the current scene graph into a specific PDDL problem file, which is then solved by a PDDL solver (Appendix A.5) to generate a subplan. Finally, all resulting subplans are concatenated to yield the complete executable plan. The underlying PDDL domain file is provided in Appendix J.

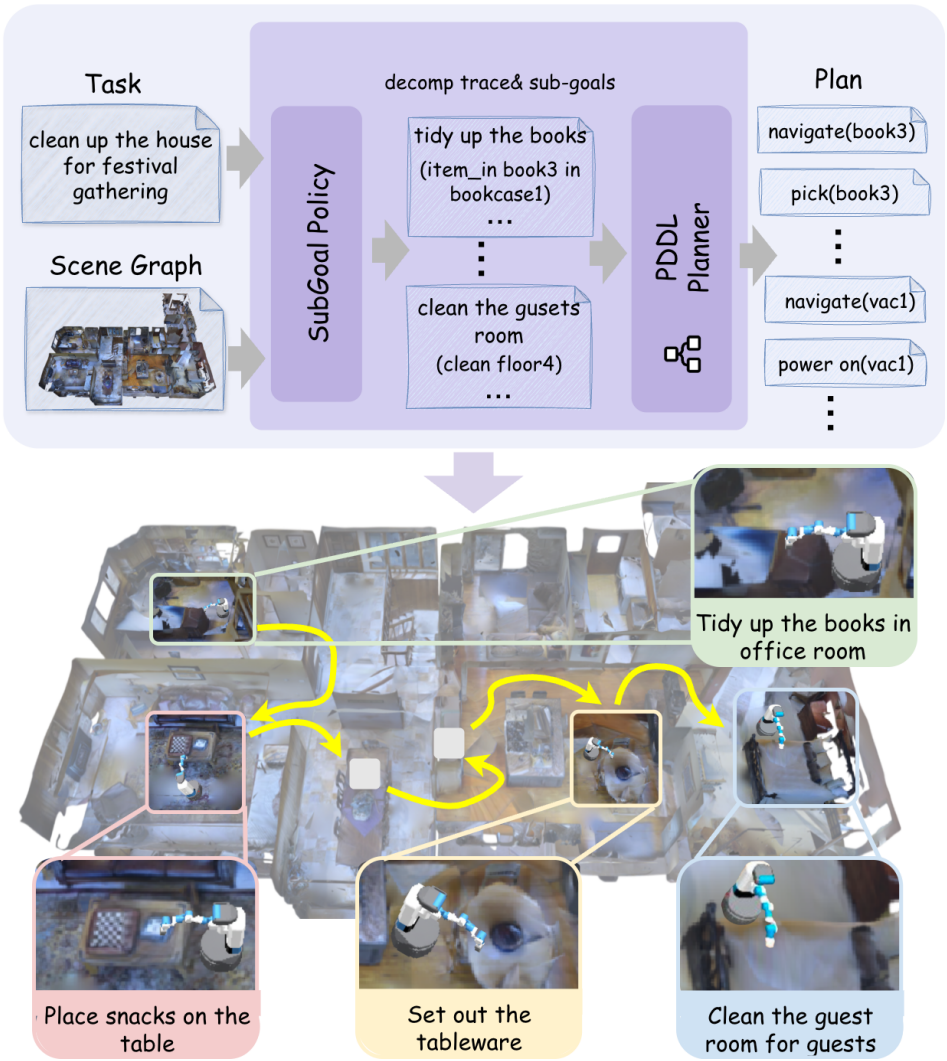


Figure A.1: In large-scale environments, the subgoal policy receives abstract instructions and a scene graph, and generates a decomposition trace with corresponding subgoals. These subgoals are then solved using a PDDL planner, resulting in an executable long-horizon plan that satisfies the user's requirements.

A.1 Scene Graph Representation and State Attributes

We formalize the environment as a scene graph (SG) comprising typed nodes and semantic edges. Specifically, the nodes represent entities categorized as `room`, `furniture`, `object`, `agent`, or `person`. The edges capture spatial relations—namely, containment (`in`) and placement (`on`).

To facilitate planning and action feasibility checking, each entity is augmented with a set of state attributes, as detailed in Table A.1. Finally, to interface with the language model, this structured SG is serialized into a textual format. A complete example of this serialized representation is provided in Listing 1.

Table A.1: State attributes stored in the scene graph for planning.

Entity type	**State attributes**

Furniture / Objects	`is_open`, `can_be_opened`
	`is_powered_on`, `is_powerable`
	`is_clean`, `is_cleaning_tool`,
	`require_floor_cleaner`, `requires_water_to_clean`
	`has_faucet`, `is_filled`
	`is_heating_device`, `is_heater`, `is_heated`

Person	`agent_at`, `p_holding`
Agent	`agent_at`, `holding`, `handempty`

Listing 1: Example of serialized textual scene graph (SG).

```
{
  "rooms": [
    {
      "name": "bathroom_1",
      "states": {
        "is_clean": false,
        "require_floor_cleaner": true
      }
    },
    {
      "name": "bedroom_1",
      "states": {
        "is_clean": false,
        "require_floor_cleaner": true
      }
    }
  ],
  "furnitures": [
    {
      "states": {
        "is_clean": false
      },
      "name": "bed_4",
      "type": "bed"
    }
  ],
  "objects": [
    {
      "states": {
        "requires_water_to_clean": true
      },
      "name": "bottle_10",
      "type": "bottle"
    },
    {
      "states": {
        "requires_water_to_clean": true
      },
      "name": "bowl_11",
      "type": "bowl"
    }
  ],
  "agent": [
```

```
    {
      "name": "agent_0",
      "location": "corridor_19"
    }
  ],
  "person": [
    {
      "name": "person_0",
      "location": "bathroom_4"
    }
  ],
  "links": [
    ["bedroom_1", "bed_4", "in"]
  ]
}
```

A.2 Decomposition Trace Output Format

We output a *decomposition trace* as an ordered list of natural-language (or structured) subtasks. Below we show a representative example.

Listing 2: Task instruction prompt used for decomposition (example).

```
Task: "Create a classy dinner atmosphere and tidy afterward"
```

Listing 3: Decomposition trace format (example).

```
### Task Decomposition ###
1. Pick_up_tablecloth_20_from_dining_table_18
2. Place_tablecloth_20_on_dining_table_18
3. Pick_up_centerpiece_84_from_dining_table_18
4. Place_centerpiece_84_next_to_tablecloth_20_on_dining_table_18
5. Pick_up_candle_141_from_dining_table_18
6. Place_candle_141_next_to_centerpiece_84_on_dining_table_18
7. Pick_up_salt_shaker_58_from_dining_table_18
8. Place_salt_shaker_58_next_to_candle_141_on_dining_table_18
9. Pick_up_pepper_grinder_59_from_dining_table_18
10. Place_pepper_grinder_59_next_to_salt_shaker_58_on_dining_table_18
11. Pick_up_candle_holder_60_from_dining_table_18
12. Place_candle_holder_60_next_to_pepper_grinder_59_on_dining_table_18
13. Pick_up_napkin_holder_22_from_dining_table_20
14. Place_napkin_holder_22_next_to_candle_holder_60_on_dining_table_18
15. Pick_up_wine_bottle_142_from_dining_table_20
16. Place_wine_bottle_142_next_to_napkin_holder_22_on_dining_table_18
```

A.3 PDDL Subgoal and Relevant Object Set Format

For each subtask t_k, the policy outputs (i) a set of grounded PDDL subgoals, and (ii) an implicitly related object set. We keep the output in a structured textual format:

Listing 4: Subgoal grounding output format (example).

```
### Subgoal Grounding ###
1. Pick_up_tablecloth_20_from_dining_table_18
   Subtask goal: ['(holding agent_0 tablecloth_20)']
   Implicitly related object: []
2. Place_tablecloth_20_on_dining_table_18
   Subtask goal: ['(item_on_surface tablecloth_20 dining_table_18)']
   Implicitly related object: []
3. Pick_up_centerpiece_84_from_dining_table_18
   Subtask goal: ['(holding agent_0 centerpiece_84)']
   Implicitly related object: []
4. Place_centerpiece_84_next_to_tablecloth_20_on_dining_table_18
```

```
   Subtask goal: ['(next_to centerpiece_84 tablecloth_20 dining_table_18)
       ↪ ']
   Implicitly related object: []
5. Pick_up_candle_141_from_dining_table_18
   Subtask goal: ['(holding agent_0 candle_141)']
   Implicitly related object: []
6. Place_candle_141_next_to_centerpiece_84_on_dining_table_18
   Subtask goal: ['(next_to candle_141 centerpiece_84 dining_table_18)']
   Implicitly related object: []
...
```

A.4 Problem file construction procedure

We formulate a subtask-specific PDDL problem instance by grounding the predicted subgoal set g_k and its related objects $objs_k$ within the current scene graph state SG_k.

Given $(SG_k, g_k, objs_k)$, the construction follows these steps:

1. **Extract core objects.** Parse all object symbols appearing in the grounded PDDL goals g_k and take their union with the implicitly related object set $objs_k$.
2. **Expand with parent nodes.** For each core object, retrieve its parent nodes within the SG hierarchy (e.g., the containing furniture and room) based on `in` and `on` spatial links.
3. **Extract initial state.** From SG_k, extract the current predicates of all collected entities, capturing spatial relations (placement/containment) and relevant state attributes.
4. **Assemble the PDDL problem.** Define the object universe using the extracted entities, instantiate the initial state with the retrieved predicates, and set the goal condition to g_k.

A.5 Subgoal Planner Details

To compute the final action sequences, we employ the off-the-shelf `Fast Downward` planner configured with the `lama-first` alias. To strictly preserve causal dependencies among subgoals, we formulate the planning process incrementally rather than solving for all goals simultaneously. Specifically, if a subtask k comprises a sequence of subgoals $g_k = \{A, B, C\}$ alongside a related object set O, we solve them sequentially as follows:

1. Formulate and solve a problem with goal A and its required objects (derived from A and O) to obtain $\texttt{subplan}_{k,A}$; execute this subplan to update SG.
2. Formulate a subsequent problem with the cumulative goal $(A \wedge B)$ and objects required for both goals to obtain $\texttt{subplan}_{k,B}$; execute and update SG.
3. Formulate the final problem with goal $(A \wedge B \wedge C)$ and all corresponding objects to obtain $\texttt{subplan}_{k,C}$.
4. Concatenate the solutions to form the complete subtask plan: $\texttt{subplan}_k = \texttt{subplan}_{k,A} \oplus \texttt{subplan}_{k,B} \oplus \texttt{subplan}_{k,C}$.

B Training Details

B.1 SFT Details

We initialize subgoal policy from a Qwen2.5-7B base model and perform supervised fine-tuning (SFT) as a warm-up stage before launching TGPO. The complete set of hyperparameters is summarized in Table B.1.

Data format. Each training instance follows the subgoal policy output schema: given an instruction q and a textual scene graph SG, the target output is a structured completion containing (i) a decomposition trace T and (ii) grounded subgoals $\{(g_k, \text{obj}_k)\}_{k=1}^{K}$.

Table B.1: SFT hyperparameters.

Hyperparameter	Value
Base model	Qwen2.5-7B
Batch size	32
SFT epochs	3
Cutoff length (`cutoff_len`)	15000
Learning rate	5×10^{-6}
Warm-up ratio	0.1

B.2 GRPO and TGPO Hyperparameters

In this section, we detail the hyperparameter configurations used for training both the baseline Group Relative Policy Optimization (GRPO) and our proposed Trace-Guided Policy Optimization (TGPO). To ensure a fair and rigorous comparison, we maintain identical settings for all shared training hyperparameters across both methods, only tuning the components specific to TGPO. The complete set of hyperparameters is summarized in Table B.2.

Table B.2: TGPO and GRPO hyperparameters.

Hyperparameter	Value
Learning rate	5×10^{-6}
Batch size	128
Mini-batch size	32
Rollout number	8
`TGPO_fix_num_per_rollout`	2

B.3 Task Completion Review Details

Task completion review is performed using Qwen3-235B-A with a single-query evaluation protocol for training efficiency. We first convert the solved plan into a detailed natural-language description, and then evaluate whether the resulting execution satisfies the task objective. In addition to the generated plan summary, the evaluator is provided with the task instruction and scene information. The model then determines whether the plan successfully completes the task. Prompt details are provided on the project page: *(View Prompt)*.

B.4 Trace Improver Details

We use Qwen3-235B-A22B-Thinking-2507 as the auxiliary model for trace refinement. During trace refinement, the model is provided with the task instruction, scene information, task-completion reviewer feedback, and the original reasoning trace. The auxiliary model revises the reasoning trace to address errors in task decomposition, temporal ordering, and intent grounding, and outputs a corrected trace for the subsequent constrained rollout stage. Prompt details are provided on the project page: *(View Prompt)*.

B.5 Training Compute Resources

All experiments are conducted on 8×H200 GPUs. Supervised fine-tuning is performed for approximately 48 hours, followed by 100 hours of reinforcement learning with TGPO.

C Rubric LLM Evaluation Details

We adopt a multi-dimensional LLM-as-judge protocol to evaluate whether a composed plan satisfies a given instruction. We decompose the evaluation into four orthogonal dimensions, query each independently with differentiated inputs, and aggregate the scores via a weighted average. This design mitigates the variance inherent in single-call evaluation.

The pipeline operates in two stages: a deterministic state computation followed by a multi-perspective LLM evaluation using GPT-5 and Gemini-3-Pro.

C.1 Deterministic State Computation via PDDL

To prevent LLM hallucination when summarizing plan accomplishments, we deterministically compute the final state by applying each action's PDDL effects to the initial scene graph $\mathcal{S}_0$ using a forward simulator. For a plan $\pi = (a_1, \ldots, a_n)$:

$$\mathcal{S}_n = \text{apply}(a_n, \text{apply}(a_{n-1}, \ldots \text{apply}(a_1, \mathcal{S}_0))) \tag{C.1}$$

The final state $\mathcal{S}_n$ is extracted as a text-formatted set of grounded predicates (e.g., `item_on_surface(plate_1, dining_table)`).

C.2 Multi-Perspective LLM Scoring

We employ GPT-5 and Gemini-3-Pro as our evaluation LLMs, querying each model four times in parallel. Each query corresponds to a distinct evaluation dimension and receives a strictly differentiated subset of information, ensuring orthogonal assessment:

Table C.1: Multi-perspective evaluation dimensions ensuring orthogonal assessment.

Dimension	**Weight**	**Input**	**Evaluation Focus & Prompt Link**
Goal Achievement	0.4	Final state + Instruction	Judges purely by outcome: whether the execution satisfies the instruction goals. *(View Prompt)*
Intent Understanding	0.2	Scene + Plan + Instruction	Enables constraint-aware evaluation: selecting the best available option demonstrates correct intent. *(View Prompt)*
Completeness	0.2	Plan + Instruction	Evaluates subtask coverage at the goal level (e.g., "set the table" counts as one covered subtask). *(View Prompt)*
Execution Quality	0.2	Plan + Instruction	Checks for unreverted state changes, self-contradictions, navigation efficiency, and action count proportionality. *(View Prompt)*

C.3 Score Aggregation

Each dimension independently outputs a discrete score $d_{i,m} \in \{0.00, 0.25, 0.50, 0.75, 1.00\}$ with a rationale. This 5-point scale further reduces variance. For model $m \in \{\text{GPT-5}, \text{Gemini-3-Pro}\}$, the score is a weighted sum:

$$s_m = \sum_{i=1}^{4} w_i \cdot d_{i,m} \tag{C.2}$$

Critical Failure Override: If D1 identifies that a core instructed object is entirely wrong, or if the plan ignores the instruction, s_m is capped at 0.2.

To ensure robustness, the final reported score is the average across both models:

$$s_{\text{final}} = \frac{s_{\text{GPT-5}} + s_{\text{Gemini-3-Pro}}}{2} \tag{C.3}$$

D Scalability Study

We conduct scalability tests to assess how TGPO scales with increasing task complexity. Fig. D.1 shows that enlarging the environment, extending the planning horizon, increasing instruction abstractness, and increasing action-constraints complexity lead to only modest performance drops for TGPO, indicating strong scalability and reasoning robustness. In contrast, baselines degrade substantially along one or more axes.

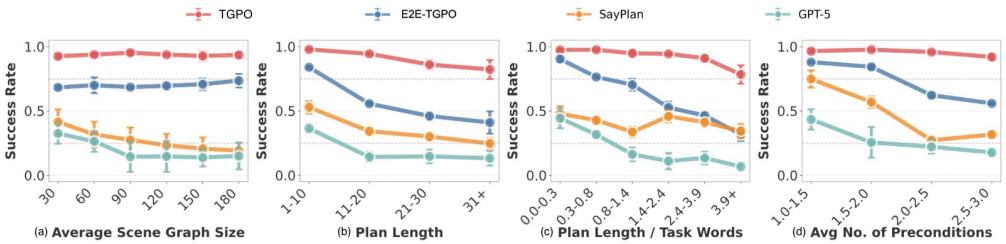


Figure D.1: Scalability Test Results showing the variation of success rate with: (a) Scene Graph Size, (b) Plan Length, (c) Task Abstractness, and (d) Constraint Complexity.

Scalability on Environment Size We measure scalability by the *average scene-graph size* (30–180 nodes). As shown in Fig. D.1(a), TGPO remains stable as graph size increases, with minimal degradation (16%). In contrast, SayPlan and GPT-5 degrade significantly, with grounding errors exacerbated by larger graphs. TGPO and E2E-TGPO are both trained with RL on our constructed dataset, whose scene graphs span a wide range of scales (50–350 nodes; avg. $\sim$150), improving robustness to large environments at test time.

Scalability on Plan Length Horizon length is scaled by plan length (1–10, 11–20, 21–30, 31+ actions). As shown in Fig. D.1(b), TGPO's performance drops by only 16% at the longest horizon, compared to 43% for E2E-TGPO. This resilience is due to TGPO 's *decompose-then-solve* structure, which prevents error accumulation by solving subplans sequentially, while E2E-TGPO suffers from cascading errors in monolithic action sequences.

Scalability on Instruction Abstractness We quantify abstractness by the ratio of plan length to instruction word count. As abstractness increases (0–3.9+), TGPO drops by 19%, while E2E-TGPO drops the most (58%). This is due to the fact that in the end-to-end setup, after TGPO trace improvement, the generated samples often fail to maintain feasibility and degrade into a GRPO-like behavior. As a result, E2E-TGPO struggles to leverage TGPO's benefits effectively, limiting its ability to handle abstract instructions.

Scalability on Constraint Complexity We measure constraint complexity by the average number of preconditions per action. As complexity increases (1.0–3.0), TGPO's performance declines by just 3%, while SayPlan (42%), E2E-TGPO (32%), and GPT-5 (26%) degrade significantly. TGPO 's symbolic planner enforces preconditions and effects, maintaining feasibility even under dense constraints, unlike purely neural generation methods.

E Low-Level Execution Details

E.1 VLA Execution

We adopt [51], which ranks first on the BEHAVIOR-1K leaderboard, as our low-level execution policy. During execution, we continuously extract environment state information from the simulator to determine high-level action termination. Specifically, the completion of each action is verified through environment state transitions. High-level action execution is controlled by stage signals provided as inputs to the low-level execution policy, enabling structured guidance over action progression.

E.2 Motion Planning

Navigation is implemented using the A* algorithm to provide reliable long-distance motion planning. To bridge the gap between classical motion planning and low-level policy execution, we adopt a hybrid strategy: A* navigation terminates when the agent reaches within a 1.5-meter radius of the target location, after which fine-grained motion is completed by the VLA model.

During navigation execution, the robot arm joints are kept fixed to avoid unnecessary motion interference, ensuring stability and consistency between navigation and manipulation phases.

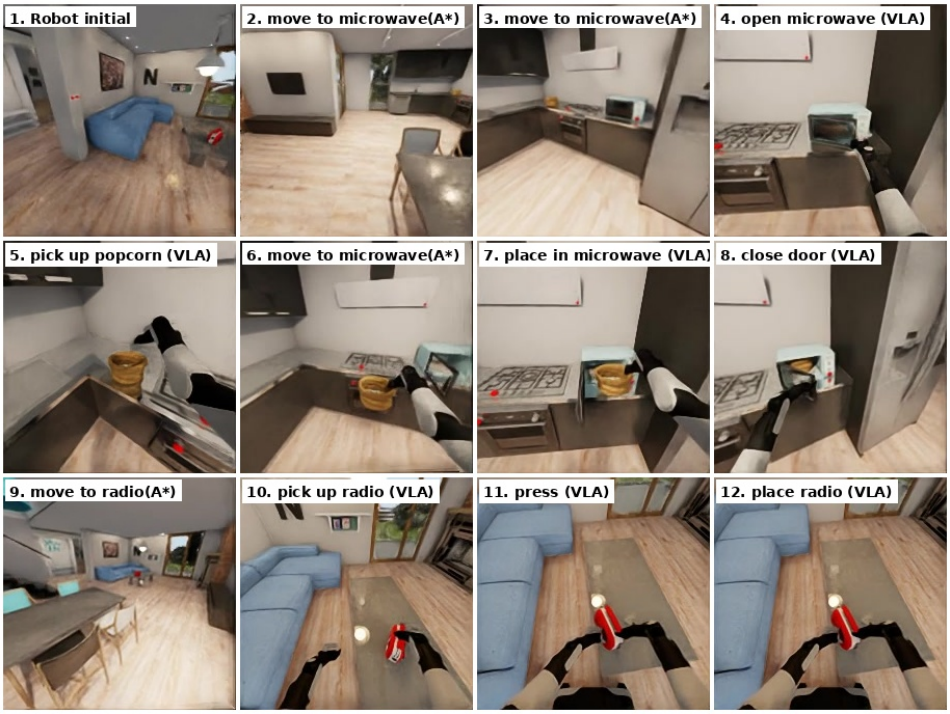


Figure E.1: Visualization of the "Preparing popcorn for a radio evening" task.

E.3 Combined Tasks

To evaluate compositional generalization, we construct combined tasks by merging partial or complete subtasks from existing tasks into new, semantically coherent household tasks. All non-navigation actions in these combined tasks are drawn from the training distribution, avoiding out-of-distribution execution by the VLA model.

The corresponding scenes are generated by merging the original scenes associated with the selected task components. This preserves environmental consistency while enabling more complex task structures. A full list of combined tasks is provided in Table E.1. We visualize an example combined task in Fig. E.1. Complete videos are provided on the project page: https://tgpo2026.github.io/TGPO/

F Benchmark Details and Analysis

F.1 Behavior Benchmark Settings

Action Space Mapping The action names in our fixed PDDL domain differ from those used in the Behavior-1K benchmark. Therefore, we construct a mapping between our action space and the Behavior-1K action space to enable consistent execution and evaluation across the two systems.

Evaluation Metrics and Success Criteria We validate plan feasibility using constraint rules extracted from the simulator for high-level planning. Task completion is evaluated based on the completion rate of the BDDL goals provided by the simulator.

F.2 In-distribution Benchmark

Examples from the in-distribution benchmark, including Easy, Complex, and Abstract tasks, are shown in Tables F.2, F.3, and F.4.

Table E.1: Combined task descriptions and their corresponding source tasks.

Combined Task	Instruction	Source Tasks
Preparing popcorn for a radio evening	Heat a popcorn bag in the microwave, then turn on the radio in the living room.	make microwave popcorn & turning on radio
Clean up kitchen after party	Collect soda cans into the trash, move pizzas to the fridge, and put bowls into the sink.	cleaning up plates and food & picking up trash
Unload groceries and store food	Bring the grocery sack from the car to the kitchen, then store items in the fridge and cabinets.	carrying in groceries & storing food
Tidy up children's room toys	Put board games, puzzles, and a ball into the toy box, then place dice, teddy bears, and games into the bookcase.	collecting children's toys & picking up toys
Prep vegetables and cook	Dice vegetables on the chopping board, then cook cabbage and chili in the frying pan.	cook cabbage & slicing vegetables
Evening relaxation setup	Bring bottles from the fridge to the coffee table, then light a fire in the fireplace.	bringing water & setting the fire
Prepare bedroom for sleep	Move the book to the nightstand and place sandals by the bed, then organize the desk.	getting organized for work & tidying bedroom
Quick lunch from fridge	Cook hot dogs in the microwave, then cook bacon in the frying pan.	cook bacon & cook hot dogs
Seasonal decoration swap	Store Halloween decorations in a cabinet, then set up Christmas decorations in the living room.	putting away Halloween decorations & putting up Christmas decorations inside
Sort vegetables and clear table	Sort vegetables into mixing bowls, then pack leftovers into containers and refrigerate.	clearing food from table into fridge & sorting vegetables

Table F.1: Action space mapping between our fixed PDDL domain and the Behavior-1K benchmark.

Ours	Behavior-1K
navigate	NAVIGATE_TO
pick_from_surface	GRASP
pick_from_receptacle	GRASP
place_on_surface	PLACE_ONTOP
place_in_receptacle	PLACE_INSIDE
open	OPEN
close	CLOSE
power_on	TOGGLE_ON
power_off	TOGGLE_OFF
pour_into	pour

F.3 Human Benchmark

We develop a web-based dialogue interface for a household service robot to collect *real* human-authored household tasks. The website visualizes the underlying scene graph to provide users with environmental context. We publicly release the survey link and explicitly inform participants that their chat logs may be used for this research. After legality and compliance screening, we obtain **50** real human household tasks. A screenshot of the dialogue website is shown in Fig. F.1. Representative human task examples, together with their task categories, are shown in Table F.5. The complete set of human-authored tasks is available on the project page: *(View Tasks)*.

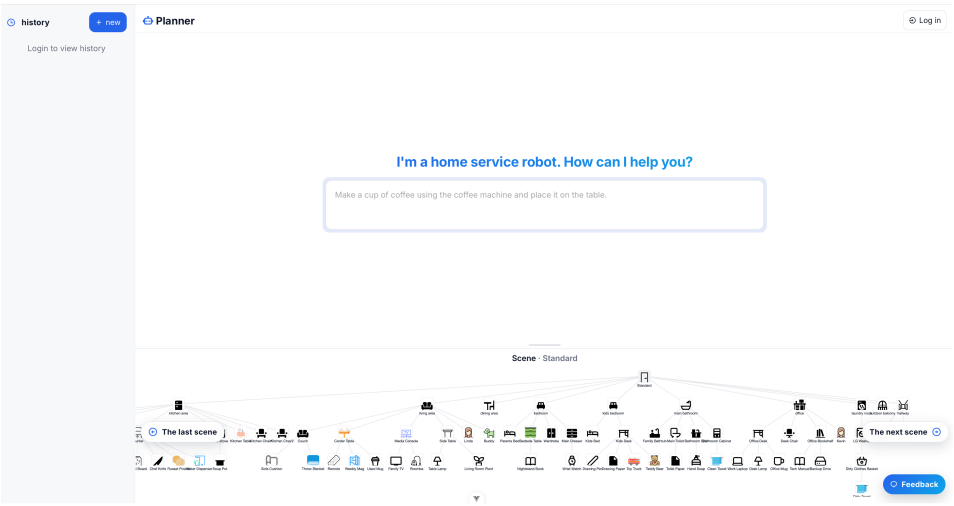


Figure F.1: Screenshot of the web-based dialogue interface used to collect real human household task instructions. The interface visualizes the scene graph to provide environmental context, and participants are informed that their conversations may be used for research purposes.

G Baseline implementation details

This section details the configuration of the baseline models used in our experiments. Specifically, we provide the exact prompt templates deployed within the baseline pipelines, alongside the specific inference budgets allocated for evaluation.

General purpose LLM details. We directly prompt the general-purpose model to generate long-horizon plans under the same input interface (instruction + scene graph).The detailed semantic search prompt and the replan prompt is provided on project page: *(View Prompt)*

SayPlan prompt. We follow the original SayPlan-style pipeline prompt, adapted to our scene-graph interface. The detailed semantic search prompt and the replan prompt are provided on the project page:*(View Prompt)*.

DELTA details. DELTA comprises three prompting steps: (i) filter the scene graph to task-relevant subgraphs, (ii) decompose the task into subtasks, and (iii) generate a PDDL problem file for each subtask. For fair comparison, we directly use our fixed PDDL domain file to solve the problem, skipping the PDDL domain file generation process of DELTA. The detailed prompt is provided at the project page: *(View Prompt)*.

LLM+P details. We use GPT-5 as the LLM and follow the original LLM+P-style prompt, adapted to our setting. The low performance is mainly caused by poor-quality problem file generation in large environments and long-horizon planning tasks. Prompt details are provided on the project page: *(View Prompt)*.

NL2Plan details For fair comparison, we skip NL2Plan's domain-file generation step and directly use our fixed PDDL domain file, while using GPT-5 as the underlying LLM. Prompt details are provided on the project page: *(View Prompt)*.

End-to-end (E2E) training details. For the end-to-end variant, we prompt the model to first decompose the task into a tree-structured reasoning trace, and then ground the decomposed subtasks into executable actions. The full training-time prompt template is provided on the project page: *(View Prompt)*.

Table F.2: Easy task examples.

ID	Instruction
E1	I'm in the bathroom — can you bring me a towel?
E2	Move a musical instrument from the living room to the bedroom.
E3	I want to do some crafts — can you bring scissors, colored paper and glue to the couch?
E4	Could you fill my reusable water bottle and give it to the person on the stairs, please?

Table F.3: Complex task examples.

ID	Instruction
C1	Refill the user's reusable water bottle and deliver it to them.
C2	Prepare a cup of boiled water and deliver it to the user.
C3	On a chilly winter morning, prepare the bathroom by pouring hot water from the kettle into a bowl placed on the bathroom counter, then bring a clean cup for rinsing.
C4	Go to the kitchen, get the mug from bench, fill it at sink, and bring the filled mug to person in bathroom and hand it over.
C5	Take a glass from the dining table, fill it with water at the kitchen sink, and hand the filled glass to person in empty_room.

Table F.4: Abstract task examples.

ID	Instruction
A1	Create a classy dinner atmosphere and tidy afterward
A2	Prepare tomorrow's morning routine station.
A3	Get ready for a cozy movie marathon.
A4	Prepare a comforting breakfast and serve it
A5	Prepare the home for guests tonight.

Table F.5: Human-authored task examples.

ID	Instruction	Category
H1	I'm thirsty. Give me a glass of water.	Explicit short-horizon
H2	I'm kind of in the mood for coffee.	Abstract long-horizon
H3	Give me some tea.	Explicit short-horizon
H4	Mop the floor and clean it all up.	Explicit short-horizon
H5	Prepare breakfast.	Abstract long-horizon

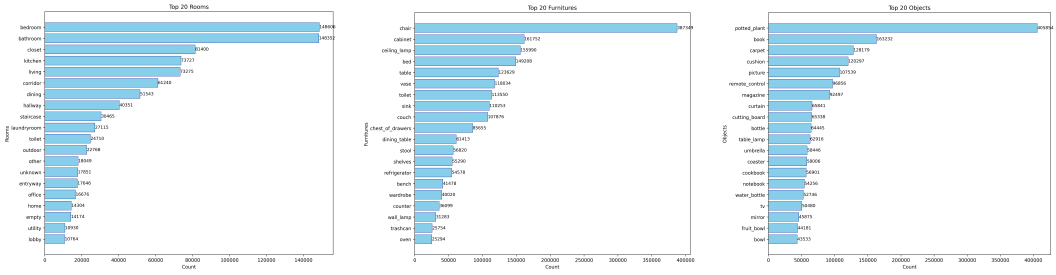
Table G.1: Baseline prompting pipelines and latency-matched inference budgets used in evaluation.

Baseline	Avg. Time (s)	Budget / Setting
GPT-5	8.0	No extra reasoning effort (latency-matched).
Gemini-3.0-pro	23.5	Low reasoning level (latency-matched).
SayPlan	63.7	Iterative prompting / replanning pipeline.
DELTA	37.8	Decompose–formulate–solve per-subtask pipeline.

H Synthetic Dataset

We summarize key statistics and provide representative examples of the scene graphs, personas, and annotated human instruction planning tasks used to construct our dataset.

Scene Graphs We construct 67.1k enriched scene instances by augmenting 308 base scene graphs with task-relevant objects and relations. Each enriched scene contains on average 142 nodes, reflecting diverse household layouts. Across the 67.1k enriched scenes, we report the top-20 most frequent **rooms**, furniture categories, and object categories in Fig. H.1, highlighting common household compositions that the planner must reason about.



(a) Top-20 rooms. (b) Top-20 furniture categories. (c) Top-20 object categories.

Figure H.1: Entity frequency statistics over the **67.1k** enriched scene instances. We visualize the **top-20** most frequent rooms, furniture categories, and object categories, respectively.

Personas To model a broad user base, we generate 1,677 distinct personas spanning diverse demographics and preferences. As illustrated in Fig. H.2, we present their age distribution, top-20 professions, and word-cloud visualizations of lifestyle tags and preference attributes. Crucially, these attributes guide our task synthesis, ensuring the generation of realistic, user-conditioned goals and constraints.

Tasks The final dataset contains 67.1k tasks that exhibit substantial structural complexity. Across the dataset, the average plan length is 10.8 steps. To quantify instruction abstractness, we define the abstraction ratio as the plan length divided by the instruction word count. Our dataset yields an average abstraction ratio of 0.76, demonstrating a high degree of implicit reasoning required. We provide representative task examples in Tables F.2–F.4, covering tasks with varying difficulty levels and degrees of abstractness. These tasks are generated using diverse prompt templates conditioned on scene graphs and sampled user personas. The complete set of task annotations is available in the released dataset.

I Additional Results and Analysis

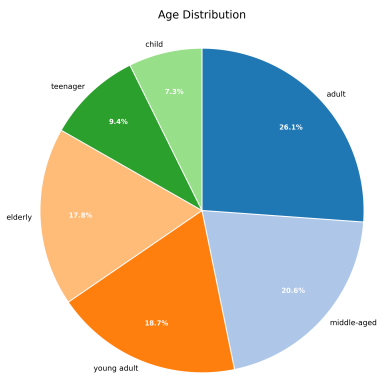
This appendix provides qualitative examples that illustrate how our method handles different instruction regimes, including *easy* (explicit), *complex* (long-horizon with multiple objectives), *abstract* (high ambiguity), and *human-authored* free-form requests. An overview of the specific tasks and their associated categories used in this analysis is summarized in Table I.1. Subsequently, for each example, we report the model reply in three parts: task decomposition, subgoal grounding, and the final PDDL action sequence.

Table I.1: Task examples used in the qualitative analysis. We report the instruction and the associated qualitative category. Detailed decomposition, grounding, and plans are shown in subsequent listings.

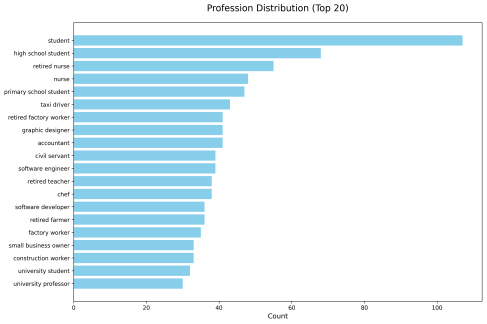
Category	Instruction
Easy	I'm in the bathroom, can you bring me a towel?
Complex	Collect scissors, tape, and colored pencils from the closet, office, and bedrooms. Set up all materials on the kitchen table for a family craft night, then power on the multiport hub in the living room so everyone can charge their devices during the activity. Afterwards, clean up by putting all used materials back in their original rooms.
Abstract	Create a classy dinner atmosphere and tidy afterward
Human Task	Give me some tea.

I.1 Easy Task Example

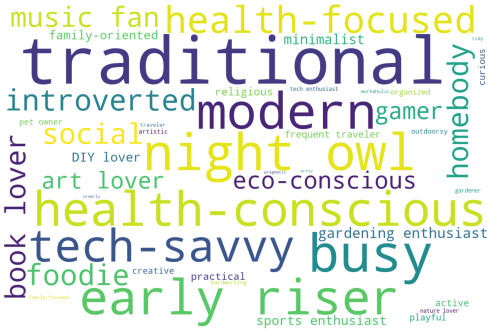
We first show an explicit request with a clear target object and destination. The output demonstrates a short decomposition with straightforward navigation, pickup, and handover. Details are provided on the project page: *(View Example)*



(a) Age-group distribution of the **1677** synthesized personas.



(b) Top-20 most frequent professions among the personas.



(c) Word cloud of lifestyle tags describing common routines and living patterns.



(d) Word cloud of preference attributes reflecting diverse user needs and constraints.

Figure H.2: Statistical overview of the synthesized personas, including age distribution, profession distribution, lifestyle tags, and preference attributes.

I.2 Complex Task Example

Next we consider a long-horizon instruction that requires (i) collecting multiple items from different rooms, (ii) arranging them on a target surface with relative placement constraints, (iii) operating an appliance, and (iv) restoring items back to their original locations. This example highlights how the decomposition maintains a coherent *setup* $\rightarrow$ *operate* $\rightarrow$ *cleanup* structure. Details are provided on project page: *(View Example)*

I.3 Abstract Task Example

We show a high-level, underspecified instruction. The reply operationalizes "classy dinner atmosphere" into a concrete table-setting procedure by selecting and arranging representative dining items. Details are provided on project page: *(View Example)*

I.4 Human Task Example

We also include a human-authored free-form request. The model decomposes the instruction into a sequence of object manipulation and appliance operations, and grounds each subtask into PDDL subgoals. Details are provided on project page: *(View Example)*

J Predefined PDDL Domain

We predefine a large-scale PDDL domain for household service robots, consisting of 28 predicates and 24 actions. Full domain details are provided on the project page: *(View PDDL Domain)*

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 clearly state the main contributions, which are consistently supported by the methods and experiments.

   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: A dedicated Limitations section (Sec. 7) is provided.

   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: [N/A]

Justification: Our paper primarily focuses on proposing a novel reinforcement learning framework and comprehensive empirical evaluations. The paper is algorithmic and empirical in nature; therefore, it does not claim or present any formal theoretical results or mathematical proofs.

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: We provide comprehensive details regarding the algorithm design (Sec. 3–4), as well as the experimental setup, baseline configurations, and evaluation protocols (Sec. 6), ensuring the reproducibility of our key results.

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: We will release anonymized code and usage instructions in the supplemental material to ensure faithful reproduction of all main results. The code is availble at: https://TGPO.github.io.

   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: The main paper describes datasets, task setups, and evaluation protocols; additional hyperparameter details are included in the appendix.

   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: We report the corresponding confidence intervals in the results Table 1 and Table 2.

   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: We detail the computational resources required to reproduce our experiments, including hardware specifications and approximate training times in Appendix B.5.

   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: We have reviewed the NeurIPS Code of Ethics, and our research complies with its guidelines; no ethical concerns are identified in this work.

   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: [N/A]

Justification: This paper focuses on foundational algorithmic improvements for embodied agent task planning and is primarily evaluated in simulated environments. As such, the work does not present immediate or foreseeable direct societal impacts (either positive or negative) that warrant a dedicated discussion.

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: The artifacts released in this work, including the synthesized dataset of robotic tasks and the trained planning policy, do not contain sensitive personal information, nor do they pose a high risk for malicious misuse that would require specialized release safeguards.

    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 external assets used in this work, including the pre-trained LLMs, robotic simulation environments, and baseline codebases, are properly cited in the paper. We have strictly adhered to their respective official licenses and usage terms as per their original releases.

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: We release the code, prompts, evaluation scripts, and processed task annotations with the camera-ready version. The released assets are accompanied by documentation describing installation, dependencies, data formats, evaluation protocols, licenses, and known limitations. No personal or sensitive human-subject data are introduced.

    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: [Yes]

    Justification: The information is provided in Appendix F.3.

    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: The paper does not involve research with human subjects and therefore does not require IRB approval.

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: LLMs play a important role in our approach. Further usage details are provided in the main paper and appendix.

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.