StructuThink: Reasoning with Task Transition Knowledge for Autonomous LLM-Based Agents

Haiyu Zhao[1], Zhenyu Guo[1], Chunhong Zhang[1], Ziyu Zhou[2], Zheng Hu[1]*
[1]State Key Laboratory of Networking and Switching Technology,
Beijing University of Posts and Telecommunications
[2]Faculty of Electrical Engineering,
Czech Technical University in Prague
{zhaohaiyu, guozhenyu2022, zhangch, huzheng}@bupt.edu.cn
zhouziyu@cvut.cz

Abstract

Decision-making tasks have highlighted fundamental challenges in grounding decisions within real-world contexts. Traditional decision knowledge utilization methods often struggle to effectively integrate structured decision constraints, limiting their ability to decompose high-level tasks, maintain logical consistency, and adapt to dynamic environments. To bridge this gap, we introduce StructuThink, a knowledge-structured reasoning framework that enhances LLM-based agents with explicit decision constraints. Specifically, we propose the Task Transition Knowledge Graph (TTKG) that learning decision knowledge in embodied scenarios. Leveraging this knowledge, we propose the StructuThink framework, comprising a subtask chain constructor for grounding natural language instructions and a constraint-based executor for adaptive and consistent decision-making. We validate StructuThink across multiple benchmarks, including ALFWorld and WebShop, where it achieves higher task success rates (improving by up to 7%) and more efficient action sequences (requiring up to 15% fewer steps) than baseline methods. Our approach enables LLMs to more effectively ground decision-making in domain-specific scenarios, enhancing both interpretability and reliability, thus paving the way for more reliable and adaptable decision-making systems.

1 Introduction

Large Language Models (LLMs) exhibit strong language understanding and generation abilities, making them promising foundations for autonomous agents in complex environments (Wang et al., 2024; Xi et al., 2025). However, their decision-making capabilities remain limited due to the lack of domain-specific training and the disconnect between high-level linguistic reasoning and low-level environmental affordances (Lu et al., 2023; Rana et al., 2023). This gap often results in failures to decompose tasks or generate context-aware actions (Xi et al., 2025; Gramopadhye and Szafir, 2023). To mitigate this, researchers have explored incorporating scene knowledge into LLMs (Song et al., 2024), which can be categorized as *Perceptual Knowledge* (e.g., object types, spatial relations) and *Apperceptive Knowledge* (e.g., commonsense, affordances, intentions). In this work, we focus on integrating the decision-related aspects of Apperceptive Knowledge to guide task execution more effectively.

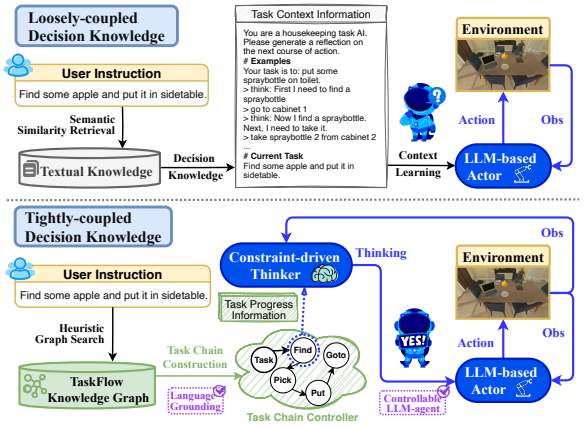


Figure 1: **Comparison of loosely- and tightly-coupled decision knowledge.** The former retrieves relevant task examples and provides pre-decision task context to guide LLMs. In contrast, our method builds a Task Transition Knowledge Graph via heuristic search and supplies real-time task progress feedback to enable constrained and adaptive decision-making.

Existing efforts to inject Apperceptive knowledge into LLMs primarily fall into two categories: unstructured and structured representations (Song et al., 2024). **Unstructured knowledge** (e.g., natural language instructions, expert demonstrations) provides flexible and diverse representations, typically learned via in-context prompting (Zhao et al., 2024; Sun et al., 2023). However, the limited con-

*Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 24489–24506
November 4-9, 2025

text window of LLMs (Xiao et al., 2024; Anokhin et al., 2024), combined with the logical vagueness of natural language, often leads to decision hallucinations (Zhu et al., 2024) and reasoning inconsistencies (Ding et al., 2023), making such methods unreliable for complex decision-making. In contrast, **structured knowledge** leverages explicit formats like logic rules or graphs to represent task states and dependencies (Anokhin et al., 2024; Zhu et al., 2024), offering better logical clarity and grounding. For example, AriGraph (Anokhin et al., 2024) constructs semantic-situational memory graphs, while KnowAgent (Zhu et al., 2024) constrains action spaces using logical rules. Yet, these approaches face a trade-off: methods like PDDL provide adaptability via real-time feedback but require costly expert-defined logic (Sun et al., 2023), whereas lower-cost tools (e.g., policy scripts, knowledge graphs) are easier to build but lack dynamic adaptation. Thus, how to efficiently incorporate structured decision knowledge into LLMs without sacrificing flexibility remains an open challenge.

Inspired by Hierarchical Task Networks (HTNs), we propose a method that utilizes subtask knowledge to construct a decision constraint graph, which imposes low-cost constraints on LLM-agent decision-making. This approach aims to bridge the gap between high-level natural language understanding and grounded task execution. To achieve this, we address the following three key questions: **Q1.** How can decision knowledge be leveraged to generate task-completable constraints from natural language instructions? **Q2.** How can LLMs be guided to accomplish tasks by adhering to constraint-based task chains? **Q3.** How can decision knowledge be generalized across scenarios by learning from expert trajectories? Specifically, our key contributions are as follows:

- We introduce a novel decision knowledge representation, termed Task Transition Knowledge Graph (TTKG), which models subtasks and their transition dependencies via graph nodes and edges (addressing **Q1**, **Q2**).
- Building on TTKG, we develop a constraint-grounded decision framework that ground natural language instructions into executable subtask chains, which are then used to dynamically guide LLM-agent decision-making (addressing **Q1**, **Q2**).
- To support scalability, we propose a automatic method for learning TTKG from expert trajectories.(addressing **Q3**)
- We empirically validate our approach in interactive environments such as ALFWorld and WebShop, demonstrating that integrating structured constraints significantly improves grounding accuracy and decision reliability.

2 Related Work

2.1 Language Models for Decision Making

Planning is a fundamental capability for intelligent agents, enabling them to decompose complex tasks into manageable sub-tasks and adapt their strategies to dynamic environments. In the context of LLM-based agents, planning typically involves two stages: formulation and reflection(Xi et al., 2025). **During plan formulation**, agents either generate comprehensive plans in a single step or adopt incremental strategies, such as the Chain-of-Thought (CoT) approach(Kojima et al., 2022; Wei et al., 2022; Lyu et al., 2023), to address sub-tasks iteratively, providing greater flexibility for intricate tasks. Hierarchical planning further enhances decision-making by organizing reasoning steps in tree-like structures(Yao et al., 2023a), allowing agents to evaluate multiple paths before finalizing a plan(Wang et al., 2022; Hao et al., 2023; Huang et al., 2022a). While Tree of Thoughts (ToT) performs tree-based reasoning over sampled "thoughts" to enhance LLM reasoning capabilities, our approach differs fundamentally by modeling subtask-level dependencies through TTKG, which provides interpretable and reusable decision knowledge specifically designed for grounded task execution. **Plan reflection** complements formulation by enabling agents to refine their strategies through internal feedback mechanisms(Shinn et al., 2023; Madaan et al., 2023; Chen et al., 2023; Miao et al., 2023), human interactions(Li et al., 2023; Wu et al., 2022; Wang et al., 2023), and environmental cues(Yao et al., 2023b; Song et al., 2023; Huang et al., 2022b; Zhao et al., 2023; Rana et al., 2023). These reflection processes help align plans with human values and improve adaptability, ensuring robust task execution in complex and real-world scenarios.

2.2 Planning Knowledge for LLM-Based Agents

Researchers have investigated diverse strategies to incorporate scene knowledge into Large Language Models (LLMs) for more effective planning and decision-making in embodied tasks. Approaches for integrating decision knowledge can be broadly categorized into unstructured and structured representations. On one hand, **unstructured representations** (e.g., natural language demonstrations or text-based experiences) (Zhao et al., 2024; Sun et al., 2023) benefit from ease of learning through in-context prompts, yet face limitations due to restricted context windows and a lack of logical rigor, often leading to "decision hallucination" in multi-step reasoning (Xiao et al., 2024; Anokhin et al., 2024; Zhu et al., 2024; Ding et al., 2023). On the other hand, **structured representations** adopt explicit formats, such as knowledge graphs (Anokhin et al., 2024) or logical constraints (Zhu et al., 2024; Roy et al.), clarifying dependencies and causal relationships for robust planning. However, achieving a balance between flexibility and construction cost remains challenging. Methods like PDDL offer dynamic adaptability but demand specialized interpreters and extensive domain expertise (Sun et al., 2023), while lower-cost alternatives (e.g., policy scripts, simplified knowledge graphs) may struggle to incorporate real-time observations for adaptive reasoning. Consequently, integrating structured planning knowledge with LLM-based processes requires careful design to maintain logical consistency and accommodate changing environmental contexts.

3 Methods

We draw inspiration from Hierarchical Task Networks (HTNs), which decompose complex tasks into structured subtasks and capture the transition relationships among them. We observe that interactive task scenarios often exhibit an inherent hierarchical structure, where the task space can be abstracted as a sequence of subtasks connected by transition dependencies. Leveraging the strong semantic understanding capabilities of LLMs, we propose to extract subtask-level decision knowledge—i.e., the subtask units and their transitions—from expert trajectories. These are stored in a Task Transition Knowledge Graph (TTKG), where nodes represent subtasks described in natural language, and edges capture learned transition knowledge between them. During decision-making, TTKG is first used to ground high-level natural language instructions into executable subtask chains. These chains then serve as dynamic constraints to guide the LLM-agent's step-by-step decisions, ensuring that its behavior aligns with task structure and prior expert knowledge.

3.1 Problem Statement for Interactive Task Execution

In our framework, the user provides a natural language instruction i, which may be long, abstract, or ambiguous. We then model the resulting complex interactive task as a sequential decision-making problem in deterministic environments. At each time step $t \in \{0, \ldots, H\}$, the agent receives an observation $o_t \in \mathcal{O}$, where $\mathcal{O}$ denotes the observation space. Based on its observation history $\mathcal{H}_t = \{o_0, o_1, \ldots, o_t\}$, the agent selects an action $a_t \in \mathcal{A}$, where $\mathcal{A}$ represents the action space. The agent's objective is to fulfill the user instruction i.

We first parse the natural language instruction into a chain of subtasks, $\mathbf{S} = \{s_1, s_2, \ldots, s_n\}$, and then generate an action sequence, $\mathbf{A} = \{a_1, a_2, \ldots, a_m\}$, based on these subtasks. The agent's decision-making is guided by a policy π, which maps the observation history $\mathcal{H}_t$ to an action a_t, i.e., $\pi : \mathcal{H}_t \rightarrow \mathcal{A}$.

3.2 Task Transition Knowledge Graph

We propose the Task Transition **Knowledge Graph (TTKG)** to represent subtask-level decision knowledge using natural language. In TTKG, both nodes and edges are expressed as natural language descriptions. Each node represents a subtask and is categorized as either a root task node (derived directly from user instructions) or a compound task node (representing intermediate goals). Edges capture transition relationships such as temporal order or dependency between subtasks. This graph structure enables the construction of interpretable subtask chains that guide LLM-agent decision-making through lightweight, language-based constraints.

3.2.1 Task Nodes

We abstract the environment's task space into three hierarchical levels: **Root Tasks** (T_r), **Compound Tasks** (T_c), and **Primitive Actions** (a). Primitive actions are the smallest executable units defined by the environment (e.g., *open drawer*, *pick up item*). Compound tasks are composed of sequences of primitive actions, i.e., $T_c = \{a^1, a^2, \ldots, a^n\}$,

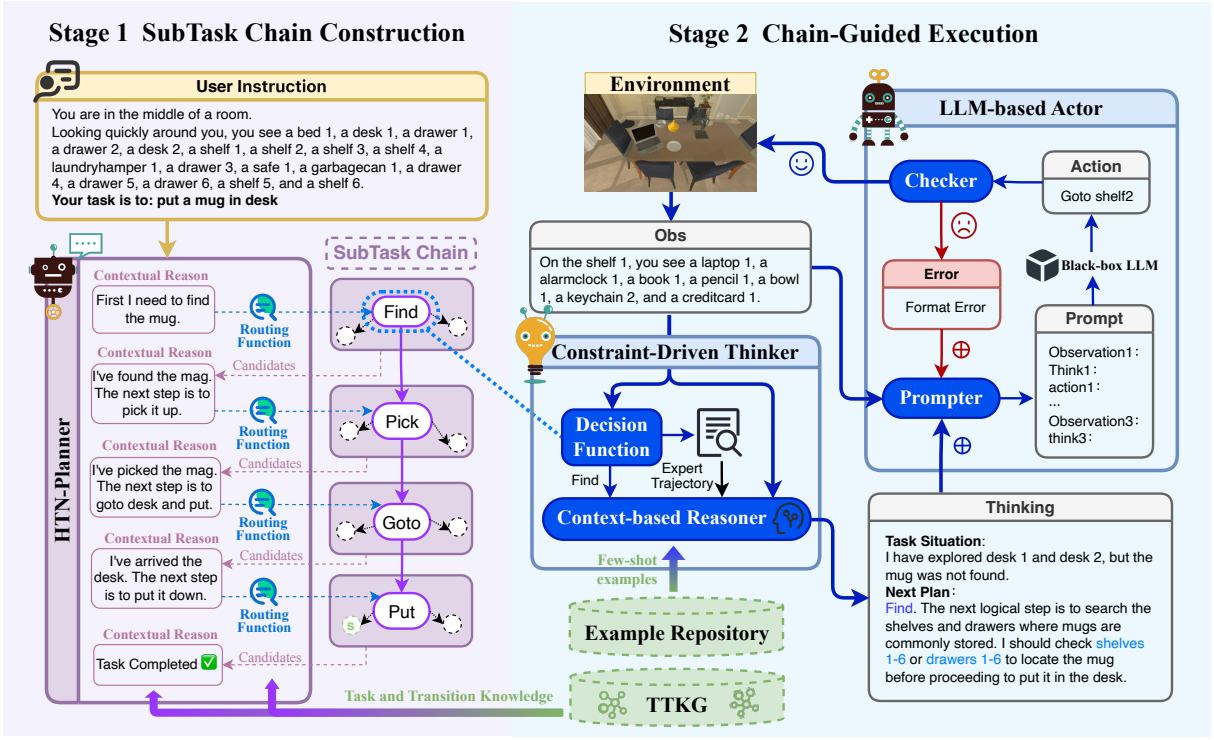


Figure 2: Overview of the **StructuThink** framework. The model operates in two stages: (1) *SubTask Chain Construction*, where the HTN-Planner decomposes user instructions into a subtask chain through a ReAct process—first reasoning over the task context, then uses the Task Transition Knowledge to select the next compound task node through a routing function; (2) *Chain-Guided Execution*, where a Constraint-Driven Thinker leverages decision functions and context-based reasoning to interpret environmental observations and guide the LLM-based actor through a prompt-reflect-act loop. Error checking and correction are performed via a checker-prompter mechanism to ensure robust execution in dynamic environments.

and serve as intermediate subgoals. Root tasks are high-level objectives derived from user instructions. In our Task Transition Knowledge Graph (TTKG), we represent only T_r and T_c as task nodes.

For each **Root Task** node T_r, we store its natural language description and a task summary outlining how it may be achieved. This information helps the system select appropriate starting tasks during compound task planning. For each **Compound Task** node T_c, we maintain a semantic description and a set of expert-provided refinements $\mathcal{R} = \{r_1, r_2, \dots\}$, where each refinement corresponds to an action sequence $\langle a_1, a_2, \dots, a_m \rangle$. The semantic description encodes the skill representation of the task, whereas the refinements serve as actionable demonstrations to guide LLM-based decision-making. Transitions between compound tasks are modeled as edges in the TTKG, which we describe in the next section.

3.2.2 Transition Edges

In TTKG, each transition edge $e = (T_i \rightarrow T_j) \in E$ is a directed link between task nodes, representing the progression from a current task T_i to its successor T_j. Since task transitions may vary across different contexts or stages, multiple edges may exist between the same node pair, each encoding a distinct decision pathway.

Each transition edge includes three key elements: (1) the identifiers of the source and target tasks T_i and T_j, (2) the **root-task information**, which contains the natural language instruction and the initial state context, and (3) a **situation summary**, which describes the current task status (e.g., "apples found but not placed"), guiding the agent's next decision. These attributes allow the LLM to reason about which transition edge to follow after completing the current compound task, thereby selecting the next subtask and supporting dynamic, step-by-step subtask generation.

3.3 StructuThink Framework

Our StructuThink framework consists of two stages: SubTask Chain Construction based on the Task Transition Knowledge Graph, and LLM-based execution with decision constraints. Dur-

ing task planning, a Root Task T_r is used to generate a **SubTask Chain** consisting of Compound Tasks. This subtask chain is typically represented as an ordered sequence, describing the dependencies among Compound Tasks. Formally, this can be represented as $C = (V, E)$, where i indicates the i-th iteration of subtask generation, $V = \{T_c^1, T_c^2, \ldots, T_c^i\}$ denotes the set of Compound Task nodes, and E represents the directed transitions between task nodes, defining their dependencies and execution order. During the execution phase, the agent interacts with the environment based on these Compound Tasks, thereby generating a sequence of Primitive Actions that accomplishes the desired objectives.

3.3.1 SubTask Chain Construction

Definition 1 (Routing Function). Given a pre-constructed knowledge graph $G = (V, E)$, a root task T_r, and a partially constructed subtask chain $C^i = [T^1, e^1, T^2, e^2, \ldots, e^{i-1}, T^i]$, the routing function f_R is defined as:

$$f_R(T_r, C^i, G) \rightarrow T_c^{i+1}$$

where $T_c^{i+1} \in V$ is the next compound task node selected based on the current context. The function leverages the information associated with the outgoing edges of the current task node T^i to determine the next task in the chain under the root task goal.

To achieve the grounding of natural language instructions into a SubTask Chain, we leverage Large Language Models (LLMs) to construct reasoning and the routing function $f_R(T_r, C^i, G)$, as outlined in Figure 2 stage 1. Specifically, in each iteration step, the model first retrieves the compound task nodes connected to the last node T_c^i in the current task chain from TTKG as candidate nodes $\mathcal{N}_{\text{cand}}$. The information of nodes and the edges between them serves as auxiliary information. Subsequently, these contexts are input into the LLM, which employs the ReAct framework to initially generate reasoning for the subsequent compound task. Following this, the model selects the next compound task T_c^{i+1} from the candidate nodes. To minimize generation costs, we prompt the LLM to perform both reasoning and node selection within a single interaction.

3.3.2 Chain-Guided Execution

Definition 2 (Decision Function). Given a compound task node T_c^i and an observation obs_t at time step t, we define a node-level decision function f_D^i as:

$$f_D^i(\text{obs}_t) \rightarrow T_c^j, \quad \text{where } T_c^j \in \{T_c^i, T_c^{i+1}\}$$

The function outputs either the current task node T_c^i (to continue execution) or the next task node T_c^{i+1} (to initiate transition), enabling dynamic and context-aware adjustment of the subtask chain during execution.

Based on the node-level decision function, we construct a decision chain framework with the ReAct paradigm. As illustrated in Figure 2 (Stage 2), execution begins by initializing the task chain C and setting the current compound task node T_c as the first node. At each time step s, the node-level decision function f_D takes the current node T_c, task chain C, and observed state o_s, and outputs whether to stay at T_c or transition to the next node. A **context-based reasoner** f_r generates a thinking trace θ_s based on T_c and o_s, which is passed to the **LLM-based actor** f_a to produce an action a_s. A **checker** module f_c validates the action; if invalid, it triggers revision via feedback until a valid action is found. This chain-guided execution loop continues until task completion or a step limit $S_{\max}$ is reached, enabling the agent to adaptively plan and act under structured constraints.

3.4 Knowledge Graph Learning

We construct the Task Transition Knowledge Graph (TTKG) from a limited set of expert trajectories by prompting a large language model (LLM) to extract and organize decision knowledge. Inspired by the work of Yang et al. (Yang et al., 2024), our TTKG focuses on encoding task-relevant decision knowledge derived from expert trajectories. The construction process is described in Algorithm 1.

Node Extraction from Expert Trajectories. From expert demonstrations, we extract three types of nodes. *Root Task Nodes* (T_r) are derived directly from natural language instructions. *Compound Task Nodes* (T_c) represent intermediate subgoals inferred from task decomposition, with each compound task node associated with an action sequence $\langle a_1, a_2, \ldots, a_n \rangle$ extracted from the demonstration. *Success Nodes* (T_{done}) denote successful task termination and serve as the endpoint of a subtask chain. Formally, for each root task T_r, we construct a subtask chain:

$$C_r = \{T_c^1, T_c^2, \ldots, T_c^n, T_{\text{done}}\}$$

Algorithm 1 TTKG Construction via Single-Shot LLM Extraction

Inputs: Expert trajectory dataset $\mathcal{D} = \{\tau_1, \ldots, \tau_M\}$, pre-trained LLM f_{LLM}
Output: Task Transition Knowledge Graph $G = (V, E)$
$G \leftarrow \varnothing$ {Initialize empty graph}
for each trajectory $\tau \in \mathcal{D}$ **do**
  /- Single LLM extraction step -/
  $(T_r, \mathcal{C}, \mathcal{E}) \leftarrow f_{\text{LLM}}(\text{prompt} = \tau)$ {T_r: root task node, $\mathcal{C}$: ordered compound task nodes, $\mathcal{E}$: transition edges}
  $V \leftarrow V \cup \{T_r\} \cup \mathcal{C}$ {Add new task nodes}
  $E \leftarrow E \cup \mathcal{E}$ {Add new transition edges}
end for
return G

where each T_c^i corresponds to a compound task derived from the trajectory and is stored in the node set V of the knowledge graph $G = (V, E)$.

Transition Edge Extraction. We extract transition edges between compound tasks based on the task execution order observed in expert demonstrations. Inspired by Zhang et al. (Zhang and Soh, 2024), we additionally encode causal knowledge about task transitions into each edge $e = (T_c^i \rightarrow T_c^{i+1})$ to capture the underlying reasons for action success or failure. In addition to the source and target nodes, each edge is annotated with three key attributes: the **root task context** from which the transition originates, the **initial observation state** prior to the transition, and a **situation summary** describing the local execution progress (e.g., "object found but not placed"). These attributes jointly capture the decision conditions under which the transition occurs, and they serve as critical inputs to the routing function f_R for guiding dynamic subtask planning.

Graph Integration and Normalization. After extracting all task nodes and transition edges, we integrate them into a unified knowledge graph $G = (V, E)$. To ensure consistency and eliminate redundancy, we apply the following integration strategies: 1) **Node normalization**, where nodes with identical semantic descriptions are merged into a single canonical node; 2) **Edge consolidation**, where multiple edges sharing the same source and target but differing in context are retained with distinct annotations to capture multiple valid transitions. The resulting TTKG supports downstream reasoning, routing, and decision-making by providing interpretable, structured decision knowledge grounded in expert behavior.

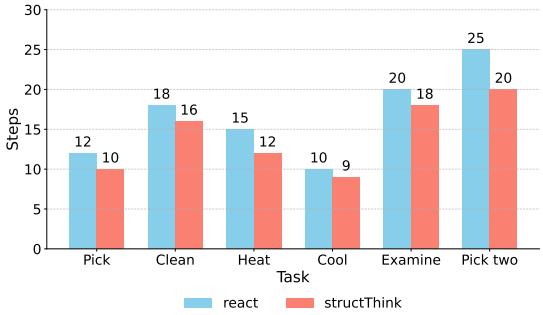


Figure 3: **Step Count Comparison Across Tasks**. This graph displays the step counts for various tasks, including only the trajectories where tasks were successfully completed. Trajectories that failed to reach a successful outcome within the maximum allowed 50 steps have been excluded.

4 Experiments and Results

4.1 Tasks, Baselines and Settings

We evaluate StructuThink on two text-based decision-making environments: ALFWorld (Shridhar et al., 2020), a virtual household environment with diverse natural language tasks (150 tasks involving object retrieval, manipulation, and navigation), and WebShop (Yao et al., 2022), a simulated online shopping platform requiring multi-step decision-making for fulfilling purchase orders (120 tasks covering various selection and transaction scenarios). Following prior work (Zhao et al., 2024; Sun et al., 2023), we use success rate as the primary evaluation metric, defined as the proportion of successfully completed episodes. In ALFWorld, episodes fail if the agent exceeds 50 actions without task completion, while in WebShop, failure is triggered by invalid actions or incomplete execution of the planned task sequence.

We compare StructuThink with several representative baselines across three strategy categories: **(1) Without Knowledge**, including ReAct (Yao et al., 2023b) and Reflexion (Shinn et al., 2023), which rely solely on in-context reasoning and self-reflection without external knowledge; **(2) Unstructured Knowledge**, including ExpeL (Zhao et al., 2024) and AdaPlanner (Sun et al., 2023), which inject unstructured external experiences or demonstrations; and **(3) Structured Knowledge**, including O3D (Xiao et al., 2024), which uses offline planning knowledge graphs. We further include ablation variants of our method:

Table 1: Performance comparison of different strategies and methods in ALFWorld and WebShop environments for *GPT-3.5-turbo* and *GPT-4-turbo*.

Strategy	Method	ALFWorld							WebShop
		Pick	Clean	Heat	Cool	Look	Pick2	ALL	
GPT-3.5-turbo Results									
Without Knowledge	React (Yao et al., 2023b)	75.0	24.7	37.7	36.4	44.4	11.8	41.9	35.0
	Reflexion (Shinn et al., 2023)	**87.5**	44.1	73.9	50.0	61.1	35.3	59.8	–
Unstructured Knowledge	ExpeL (Zhao et al., 2024)	62.5	61.3	30.4	61.9	55.5	35.3	52.2	41.0
	AdaPlanner (Sun et al., 2023)	83.3	46.2	65.2	**74.2**	68.5	52.9	63.3	–
Structured Knowledge	O3D (Xiao et al., 2024)	71.2	35.4	4.1	66.8	43.7	24.3	41.0	35.1
	Ours(woStruct)	50.2	58.1	52.9	53.4	58.6	56.7	56.3	37.6
	Ours	81.4	**75.2**	**83.1**	68.3	**73.6**	**75.0**	**76.2**	**53.4**
GPT-4-turbo Results									
Without Knowledge	React (Yao et al., 2023b)	95.8	76.3	69.6	86.4	72.2	52.9	76.8	40.0
	Reflexion (Shinn et al., 2023)	**100.0**	95.7	78.3	86.4	77.8	70.6	85.9	–
Unstructured Knowledge	ExpeL (Zhao et al., 2024)	94.4	82.8	72.4	81.8	72.2	58.8	79.2	45.0
	AdaPlanner (Sun et al., 2023)	88.9	90.3	85.5	75.8	64.8	41.2	76.4	–
Structured Knowledge	O3D (Xiao et al., 2024)	92.3	**99.7**	**96.4**	**94.8**	**99.5**	53.1	91.2	57.3
	Ours(woStruct)	84.3	86.2	83.5	85.0	87.1	83.7	84.3	42.9
	Ours	93.1	92.3	91.8	91.9	93.7	**92.6**	**92.0**	**59.1**

Ours(woStruct), which removes structured decision constraints from TTKG, and Ours, the full StructuThink pipeline.

4.2 Main Results

Table 1 shows that StructuThink achieves the best performance across both ALFWorld and WebShop. It outperforms unstructured (e.g., AdaPlanner) and prior structured methods (e.g., O3D), with success rates of 76.2% (GPT-3.5) and 92.0% (GPT-4) in ALFWorld, and 53.4%/59.1% in WebShop. Gains are especially prominent on complex tasks like *Pick2* and *Heat*, where task dependencies matter. Notably, StructuThink shows clear advantages even under GPT-3.5, validating its robustness in low-resource LLM settings.

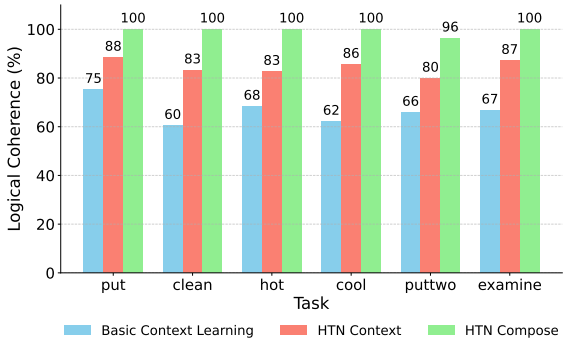
Step Count Analysis. Figure 3 shows that StructuThink achieves shorter trajectories compared to baselines, indicating more efficient task execution. In ALFWorld, it reduces steps by 17.4% over AdaPlanner and 23.5% over ReAct (Yao et al., 2023b). Similar gains are observed in WebShop. These improvements stem from the structured constraints guiding agents to relevant subgoals, minimizing redundant actions and revisions.

Error Analysis. Unstructured methods often suffer from decision hallucinations and inefficient exploration, especially in tasks like *Look* or *Pick2*, where reasoning dependencies are crucial. Approaches such as Reflexion (Shinn et al., 2023) rely heavily on iterative refinements, leading to increased trial-and-error. In contrast, StructuThink enforces valid subtask ordering through explicit task graphs, reducing failures due to misordered or illogical decisions.

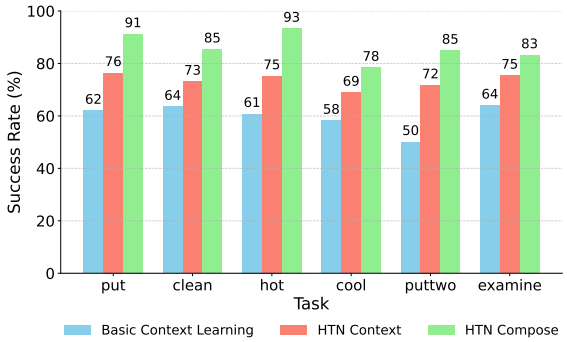
4.3 Task Chain Analysis

We conduct a task chain analysis to examine how effectively each method maintains logical consistency when decomposing a given instruction into subtasks. Specifically, we compare three different settings: (1) *Basic Context*, where we only provide the original instruction and a high-level task decomposition requirement; (2) *HTN-Knowledge Prompt*, where we additionally include an HTN-based knowledge graph in a prompt-like format; and (3) *HTN Compose*, which applies a hierarchical task network algorithm layer by layer using the HTN knowledge graph. Our main **metrics** are *Logical Coherence*, reflecting whether the produced subtasks follow valid prerequisite/action-order constraints, and *Success Rate*, indicating how effectively the generated task chain performs in the actual execution of the task.

As shown in Figure 4a and 4b, integrating HTN structures significantly improves both logical coherence and execution success. The *Basic Context* setting often yields invalid subtask sequences (e.g., missing prerequisites), leading to low coherence (e.g., 62.1% on *cool*) and poor completion.



(a) Logical Coherence



(b) Success Rate

Figure 4: **Comparison of logical coherence and success rates for different methods** on the put, clean, hot, cool, puttwo, and examine tasks. Higher percentages indicate more consistent subtask ordering, fewer prerequisite violations, and a greater likelihood of completing the subtask sequence without errors.

HTN-Knowledge Prompt improves coherence by 10–20 points, highlighting the benefit of exposing task dependencies. However, without structural enforcement, ordering errors persist. In contrast, *HTN Compose* achieves the highest performance by composing tasks layer-by-layer with graph-based validation, ensuring valid subtask transitions and robust execution.

4.4 Learning Efficiency Comparison

To assess the effectiveness and efficiency of structured knowledge integration, we compare our method (Ours) and its unstructured variant (Ours(woStruct)) against the React baseline under varying numbers of expert demonstrations (6–30) in the ALFWorld environment using GPT-3.5. This setup evaluates how well each method generalizes task execution as demonstration data increases.

As shown in Figure 5, our structured method consistently outperforms both baselines, especially in low-data settings. With only 12 demonstrations, it stabilizes at an 85% success rate, while the unstructured variant requires over 18 demonstrations to approach 75%, and React remains below 50% even with 24 demonstrations. These results highlight the superior sample efficiency and generalization of our structured approach, confirming the value of explicit task knowledge in guiding LLM-based agents.

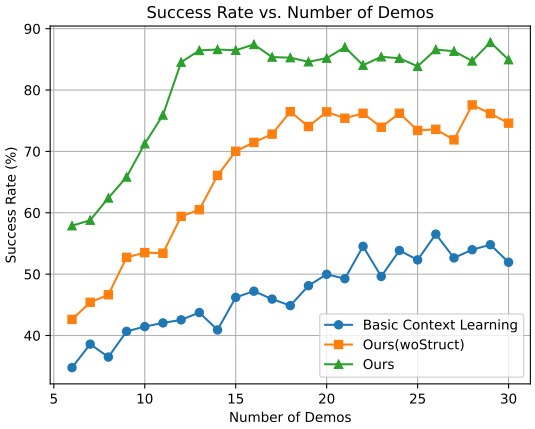


Figure 5: **Success rate curves of *GPT-3.5*-based experiments**. We compare our proposed method (Ours) and its variant without structural constraints (Ours(woStruct)) against the React method. As the number of expert trajectories increases, our approaches achieve higher and more stable success rates than React.

4.5 Ablation Studies

To evaluate the contributions of individual components in our structured reasoning framework, we conduct ablation studies by selectively removing key modules: (1) Expert Trajectory, which provides reference demonstrations for task execution; (2) Checker, a validation mechanism ensuring consistency in decision-making; and (3) Struct, our structured decision constraint mechanism. The results, summarized in Table 2, highlight the necessity of each component in enhancing task performance.

As shown in Table 2, removing any component leads to a notable performance drop, confirming their complementary contributions. Excluding **Struct** reduces success to 76.4%, highlighting its role in maintaining logical consistency. Without the **Checker**, success drops to 80.2%, indicating the benefit of runtime validation. The largest decline occurs without **Expert Trajectories** (73.7%), demonstrating their importance in guiding LLMs via demonstration-based grounding. Overall, the full model achieves the best performance (86.3%), validating the effectiveness of our structured reasoning framework.

Table 2: **Ablation study** on task performance in the ALFWorld dataset.

Expert Trajectory	Checker	Struct	Success Rate (%)
✓	✓		76.4
✓		✓	80.2
	✓	✓	73.7
✓	✓	✓	**86.3**

5 Conclusions

We propose **StructuThink**, a knowledge-structured reasoning framework that enhances the reliability and interpretability of LLM-based decision-making in embodied environments. By introducing the *Task Transition Knowledge Graph* (TTKG) and a heuristic grounding algorithm, StructuThink effectively bridges natural language instructions and executable task chains, enabling logically consistent, context-aware planning. Experiments on ALFWorld and WebShop show significant gains in task success and efficiency over state-of-the-art baselines. Our method improves grounding, modularity, and robustness without domain-specific training, offering a scalable path toward more transparent and adaptive autonomous agents.

Limitations

While our proposed StructuThink framework demonstrates strong performance in text-based embodied decision-making environments, it also has several limitations that open avenues for future work.

Limited modality and environment scope. Our current evaluation is restricted to language-based decision-making benchmarks, such as ALFWorld and WebShop, which, although interactive, abstract away many complexities of real-world multimodal environments. Consequently, the effectiveness of our approach in more realistic embodied settings—such as those involving visual perception, physical manipulation, or real-time robot control—remains unverified. We plan to extend StructuThink to multimodal embodied agents by integrating visual grounding and sensory feedback, allowing agents to operate in physical or simulated 3D environments with richer contextual cues.

Dependence on pretrained LLM capabilities. The effectiveness of StructuThink heavily relies on the reasoning and generalization capabilities of large pretrained language models. While LLMs offer powerful linguistic and commonsense priors, their planning quality can degrade in tasks requiring domain-specific knowledge or long-horizon reasoning. Moreover, without task-specific tuning, LLMs may struggle to leverage the full structure encoded in the TTKG. To mitigate this, future work could explore fine-tuning LLMs using trajectories aligned with the TTKG structure, enabling better grounding of planning decisions in graph-based task knowledge and enhancing the overall consistency and accuracy of the agent's behavior.

References

Petr Anokhin, Nikita Semenov, Artyom Sorokin, Dmitry Evseev, Mikhail Burtsev, and Evgeny Burnaev. 2024. Arigraph: Learning knowledge graph world models with episodic memory for llm agents. *arXiv preprint arXiv:2407.04363*.

Zhipeng Chen, Kun Zhou, Beichen Zhang, Zheng Gong, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Chatcot: Tool-augmented chain-of-thought reasoning on chat-based large language models. *arXiv preprint arXiv:2305.14323*.

Yan Ding, Xiaohan Zhang, Saeid Amiri, Nieqing Cao, Hao Yang, Andy Kaminski, Chad Esselink, and Shiqi Zhang. 2023. Integrating action knowledge and llms for task planning and situation handling in open worlds. *Autonomous Robots*, 47(8):981–997.

Maitrey Gramopadhye and Daniel Szafir. 2023. Generating executable action plans with environmentally-aware language models. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3568–3575. IEEE.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Hong, Zhen Wang, Daisy Wang, and Zhiting Hu. 2023. Reasoning with Language Model is Planning with World Model. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 8154–8173. Association for Computational Linguistics.

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

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, and 1 others. 2022b. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in*

neural information processing systems, 35:22199–22213.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in Neural Information Processing Systems*, 36:51991–52008.

Guanxing Lu, Ziwei Wang, Changliu Liu, Jiwen Lu, and Yansong Tang. 2023. Thinkbot: Embodied instruction following with thought chain reasoning. *ArXiv*, abs/2312.07062.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful Chain-of-Thought Reasoning. In *Proceedings of the 13th International Joint Conference on Natural Language Processing and the 3rd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 305–329. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *ArXiv*, abs/2303.17651.

Ning Miao, Yee Whye Teh, and Tom Rainforth. 2023. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning. *arXiv preprint arXiv:2308.00436*.

Krishan Rana, Jesse Haviland, Sourav Garg, Jad Abou-Chakra, Ian Reid, and Niko Suenderhauf. 2023. SayPlan: Grounding Large Language Models using 3D Scene Graphs for Scalable Robot Task Planning. In *Proceedings of The 7th Conference on Robot Learning*, pages 23–72. PMLR.

Shamik Roy, Sailik Sengupta, Daniele Bonadiman, Saab Mansour, and Arshit Gupta. FLAP: Flow-Adhering Planning with Constrained Decoding in LLMs. arXiv.

Noah Shinn, Federico Cassano, Beck Labash, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Neural Information Processing Systems*.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew J. Hausknecht. 2020. Alfworld: Aligning text and embodied environments for interactive learning. *ArXiv*, abs/2010.03768.

Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2023. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2998–3009.

Yaoxian Song, Penglei Sun, Haoyu Liu, Zhixu Li, Wei Song, Yanghua Xiao, and Xiaofang Zhou. 2024. Scene-Driven Multimodal Knowledge Graph Construction for Embodied AI. *IEEE Transactions on Knowledge and Data Engineering*, 36(11):6962–6976.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. 2023. AdaPlanner: Adaptive Planning from Feedback with Language Models. In *NIPS '23: Proceedings of the 37th International Conference on Neural Information Processing Systems*.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. *Transactions on Machine Learning Research*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-Consistency Improves Chain of Thought Reasoning in Language Models. In *ICLR 2023*. arXiv.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Advances in Neural Information Processing Systems*.

Tongshuang Wu, Michael Terry, and Carrie Jun Cai. 2022. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, pages 1–22. Association for Computing Machinery.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, and 1 others. 2025. The rise and potential of large language model based agents: A survey. *Science China Information Sciences*, 68(2):121101.

Yuchen Xiao, Yanchao Sun, Mengda Xu, Udari Madhushani, Jared Vann, Deepeka Garg, and Sumitra Ganesh. 2024. O3D: Offline Data-driven Discovery and Distillation for Sequential Decision-Making with Large Language Models. In *COLM*. arXiv.

Rui Yang, Boming Yang, Aosong Feng, Sixun Ouyang, Moritz Blum, Tianwei She, Yuang Jiang, Freddy Lecue, Jinghui Lu, and Irene Li. 2024. Graphusion: a rag framework for knowledge graph construction with a global perspective. *arXiv preprint arXiv:2410.17600*.

Shunyu Yao, Howard Chen, John Yang, and Karthik Narasimhan. 2022. WebShop: Towards Scalable Real-World Web Interaction with Grounded Language Agents. *Advances in Neural Information Processing Systems*, 35:20744–20757.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R. Narasimhan. 2023a. Tree of Thoughts: Deliberate Problem Solving with Large Language Models. In *Thirty-Seventh Conference on Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023b. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*. arXiv.

Bowen Zhang and Harold Soh. 2024. Extract, define, canonicalize: An llm-based framework for knowledge graph construction. *arXiv preprint arXiv:2404.03868*.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. 2024. ExpeL: LLM Agents Are Experiential Learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19632–19642.

Xufeng Zhao, Mengdi Li, Cornelius Weber, Muhammad Burhan Hafez, and Stefan Wermter. 2023. Chat with the environment: Interactive multimodal perception using large language models. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3590–3596. IEEE.

Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Ningyu Zhang, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, and Huajun Chen. 2024. KnowAgent: Knowledge-Augmented Planning for LLM-Based Agents. In *NAACL 2025*.

A Prompts Used in Experiments

This appendix presents the prompts used during our experiments to guide the large language model. These prompts fall into three categories:

- **Prompt A: First-Node Selection** — used to select the initial task node on a given path (see Appendix A.1).
- **Prompt B: Next-Node Selection** — used to select the subsequent task node based on the current path state (see Appendix A.2).
- **Prompt C: Chain-Guided Execution** — used to guide the model's reasoning and final output generation (see Appendix A.3).
- **Prompt D: TTKG Learning** — used to learn the TTKG from the expert trajectories (see Appendix A.4).

These prompts serve as key components of our approach, enabling the model to reason and generate actions effectively across complex task sequences.

A.1 Prompt A: SubTask Chain Construction for First Root Task Node

Prompt Example: SubTask Chain Construction for First Root Task Node

Please select what kind of task the current instruction is according to the following candidate tasks.

- **Task Name:** `pick_and_place`, **Description:** put some spraybottle on toilet.
- **Task Name:** `look_at_obj`, **Description:** look at bowl under the desklamp.
- **Task Name:** `pick_clean_then_place`, **Description:** put a clean lettuce in diningtable.
- **Task Name:** `pick_heat_then_place`, **Description:** put a hot apple in fridge.
- **Task Name:** `pick_cool_then_place`, **Description:** cool some potato and put it in diningtable.
- **Task Name:** `pick_two_obj`, **Description:** put two creditcard in dresser.

Current task instruction: heat some apple and put it in fridge
Please only output the task name of the task.

A.2 Prompt B: SubTask Chain Construction for Next Root Task Node

Prompt Example: SubTask Chain Construction for Next Root Task Node

Please output the next subtask to be completed according to the current task and the currently completed subtasks. Refer to the experience of completing the current task in the previous tasks.

Some experiences after completing the current subtask in the previous task:

- Task: put two creditcard in dresser., Next Subtask: Pick, Experience: The main task is to put two credit cards in the dresser, and the current execution state is "Now I find two credit cards," so, next I need to pick them up in order to proceed with placing them in the dresser.
- Task: cool some potato and put it in diningtable., Next Subtask: Pick, Experience: The main task is to cool some potato and put it in the dining table, and the current execution state is "Now I find a potato," so, next I need to pick it up to proceed with cooling it.
- Task: put a hot apple in fridge., Next Subtask: Pick, Experience: The main task is to put a hot apple in the fridge, and the current execution state is "Now I find an apple," so, next I need to pick it up to proceed with heating it before placing it in the fridge.
- Task: put a clean lettuce in diningtable., Next Subtask: Pick, Experience: The main task is to put a clean lettuce on the dining table, and the current execution state is "Now I find a lettuce," so, next I need to pick it up to proceed with cleaning and placing it on the dining table.
- Task: look at bowl under the desklamp., Next Subtask: Pick, Experience: The main task is to look at the bowl under the desklamp, and the current execution state is "Now I find the bowl," so, next I need to pick it up to get a closer look.
- Task: put some spraybottle on toilet., Next Subtask: Pick, Experience: Now I find a spraybottle (2). Next, I need to take it.
- Task: put two creditcard in dresser., Next Subtask: Pick, Experience: The main task is to put two credit cards in the dresser, and the current execution state is "Now I find a credit card," so, next I need to pick it up to proceed with placing it in the dresser.
- Task: look at bowl under the desklamp., Next Subtask: toggle on, Experience: The main task is to look at the bowl under the desklamp, and the current execution state is "Now I have found the bowl and the desklamp," so, next I need to toggle on the desklamp to illuminate the bowl for better visibility.

Output Instructions:

If the task has not been completed, please output the next subtask, selecting from `Pick, Put, Find, Navigation, toggle on, Cool, Clean, Heat`.
If the task has been completed, please output `Mession Success`.
Please only output the subtask you have selected or `Mession Success`.

Current Task:

Task: `heat some apple and put it in fridge`
The subtasks that have been completed currently: `pick_heat_then_place, Find`
Please only output the subtask name you have selected or `Mession Success`.

A.3 Prompt C: Chain-Guided Execution

Prompt Example: Chain-Guided Execution in ALFWorld

Task Description

You are a household assistant that must complete the task below inside an ALFWorld environment. The following is an explanation of the current task to be executed, the subtasks required to execute this task, as well as the task currently being executed. Please analyze the current task execution status based on the information below, and output the next action plan as required.

—

 # Action List

- `go to {somewhere}`, example: `go to sidetable 2`
- `open {object}`, example: `open fridge 1`
- `close {object}`, example: `close cabinet 2`
- `take {object1} from {object2}`, example: `take spraybottle 2 from cabinet 2`
- `put {object1} in/on {object2}`, example: `put egg 2 in/on diningtable 1`
- `heat {object1} with {object2}`, example: `heat egg 2 with microwave 1`
- `cool {object1} with {object2}`, example: `cool pan 1 with fridge 1`
- `clean {object1} with {object2}`, example: `clean lettuce 1 with sinkbasin 1`
- `use {object}`, example: `use desklamp 1`

—

 # Task and Subtask Explanation

- **Task:** `heat some apple and put it in fridge`
- **Subtasks:**
  - **Find**, Description: None.
  - **Navigation**, Description: None.
  - **Put**, Description: None.

—

 # Interaction History

`Obs:` You are in the middle of a room. Looking quickly around you, you see a fridge 1, a cabinet 1, a countertop 1, a toaster 1, a coffeemachine 1, a countertop 2, a cabinet 2, a stoveburner 1, a stoveburner 2, a cabinet 3, a cabinet 4, a microwave 1, a countertop 3, a sink 1, a sink 2, a shelf 1, a shelf 2, a drawer 1, a drawer 2, a drawer 3, a shelf 3, a stoveburner 3, a stoveburner 4, and a garbagecan 1.
`Your task is to: heat some apple and put it in fridge`

—

 # Current Sub-task

`Find`

—

 # Output Requirements

- **Thoughts on subtask status:** Output thoughts on the current subtask execution status

Prompt Example: Chain-Guided Execution in ALFWorld (Continued)

- **Current Subtask:** Only output a subtask name. If the current subtask is not completed, continue with the current subtask; otherwise, select the next subtask.
- **Thoughts on the next action:** Output thoughts on the next action
- **Next action:** Output the next action

—

 # Output Format (MUST be valid JSON)

```
{
    "Thoughts on subtask status": "...",
    "Current Subtask": "...",
    "Thoughts on the next action": "...",
    "Next action": "..."
}
```

Prompt Example: Chain-Guided Execution in WebShop

Task Description

You are an online shopping assistant that must complete the task below inside a WebShop environment. The following is an explanation of the current task to be executed, the subtasks required to fulfill this task, as well as the current subtask being executed. Please analyze the task execution status based on the interaction history and output the next action plan in the required format.

—

 # Action List

- `search[{query}]`, example: `search[wireless mouse under 20 dollars]`
- `click[{product_id}]`, example: `click[B078GWRC1J]`
- `click[{option}]`, example: `click[3 ounce (pack of 1)]`
- `think[{thoughts}]`, example: `think[item matches all criteria, proceed to checkout]`
- `click[Buy Now]`, example: `click[Buy Now]`

—

 # Task and Subtask Explanation

- **Task:** `Find and buy a 3 ounce bottle of bright citrus deodorant for sensitive skin, price under 50 dollars`
- **Subtasks:**
  - **Search**, Description: Locate candidate items using relevant queries.
  - **Select**, Description: Inspect and verify item attributes to ensure requirements are met.
  - **Buy**, Description: Choose options and finalize the purchase.

Prompt Example: Chain-Guided Execution in WebShop (Continued)

—

 # Interaction History

`Instruction:` i would like a 3 ounce bottle of bright citrus deodorant for sensitive skin, and price lower than 50.00 dollars
`Action:` search[3 ounce bright citrus deodorant sensitive skin]
`Observation:` [B078GWRC1J] Bright Citrus Deodorant, 3-Ounce, 10.99; $[B078GTKVXY]GingerFreshDeodorant, 3-Ounce$,10.99
`Action:` think[B078GWRC1J and B078GTKVXY match requirements. Check B078GWRC1J first.]
`Observation:` OK.
`Action:` click[B078GWRC1J]
`Observation:` Product page with scent and size options
`Action:` click[bright citrus]
`Observation:` You have clicked bright citrus.
`Action:` click[3 ounce (pack of 1)]
`Observation:` You have clicked 3 ounce (pack of 1).

—

 # Current Sub-task

`Buy`

—

 # Output Requirements

- **Thoughts on subtask status:** Output thoughts on the current subtask execution status
- **Current Subtask:** Only output a subtask name. If the current subtask is not completed, continue with the current subtask; otherwise, select the next subtask.
- **Thoughts on the next action:** Output thoughts on the next action
- **Next action:** Output the next action

—

 # Output Format (MUST be valid JSON)

```
{
    "Thoughts on subtask status": "...",
    "Current Subtask": "...",
    "Thoughts on the next action": "...",
    "Next action": "..."
}
```

A.4 Prompt D: TTKG Learning

Prompt Example: Chain-Guided Execution in WebShop (Continued)

You are an AI agent that converts embodied-AI expert trajectories into structured JSON describing a decision-knowledge graph.
Please split the tasks into several categories of subtasks according to the expert's track record, and return the results in the required format.
Follow the schema *exactly*; do **not** invent extra keys.
Here is an expert trajectory delimited by triple backticks:
“‘`trajectory`“‘
Return **pure JSON** (no markdown) with this schema:

```
{
  "task": {
    "name": str,
    "task_description": str,
    "init_obs": str,
    "think_first_plan": str
  },
  "subtasks": [
    {
      "name": str,
      "subtask_description": str,
      "movement": str,
      "think(edge_reasoning)": str
    },
    ...
  ]
}
```

Ensure array order matches execution order.

B TTKG Learning Examples

This section provides illustrative examples of the Task Transition Knowledge Graph (TTKG) learned from expert trajectories. The following two figures present typical subtask chains and transition structures extracted by our method, demonstrating how task knowledge is organized and utilized within the StructuThink framework.

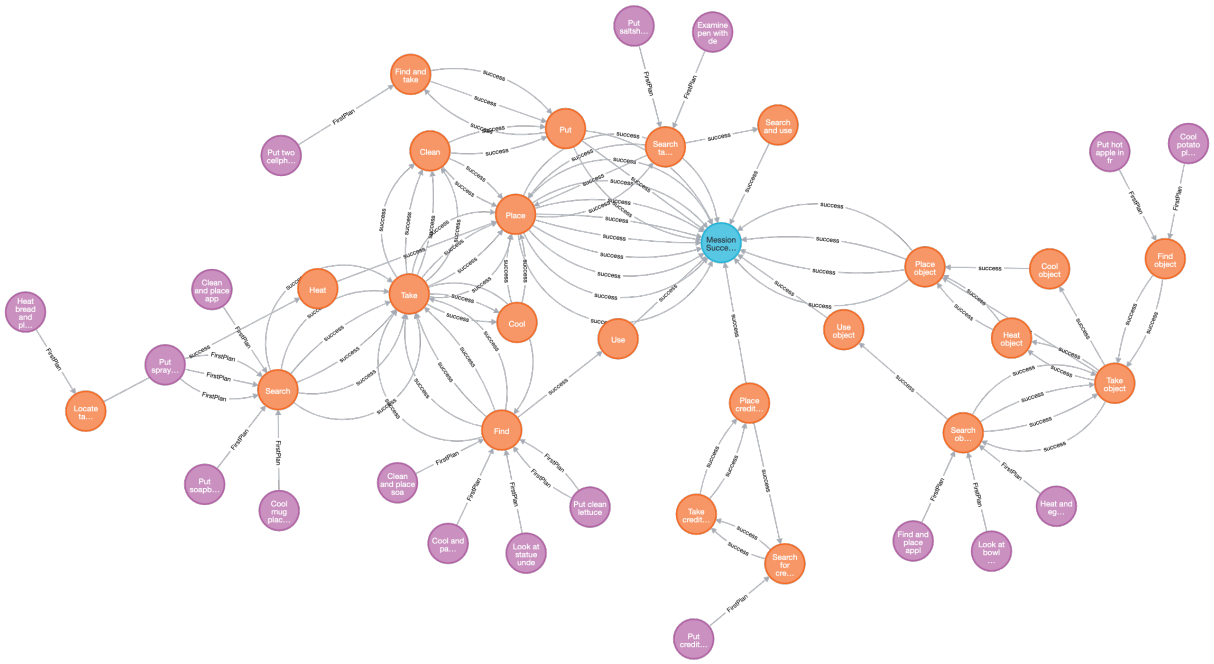


Figure 6: TTKG Example: ALFWorld

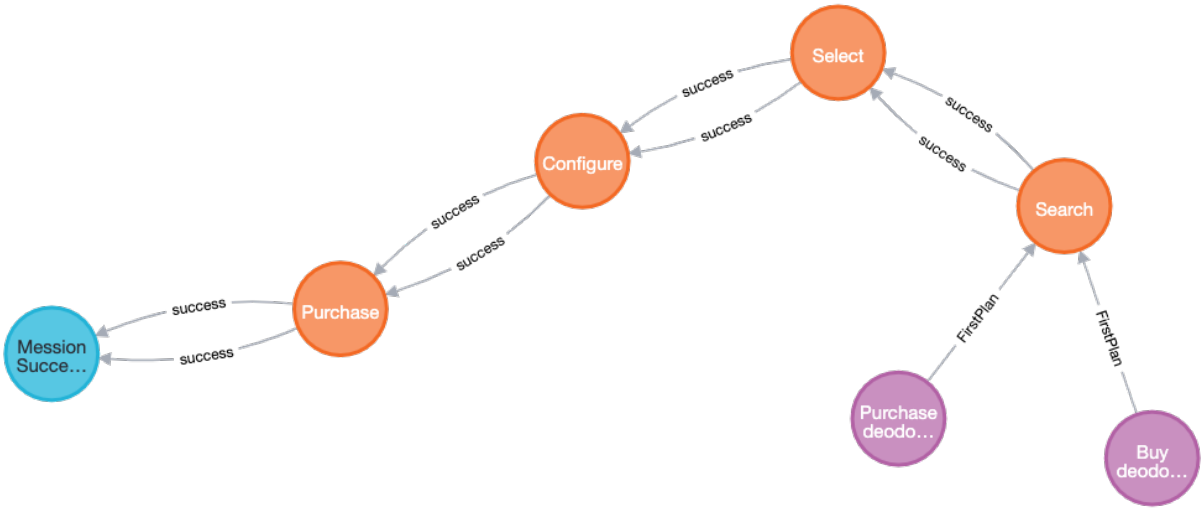


Figure 7: TTKG Example: WebShop