Integrating Intent Understanding and Optimal Behavior Planning for Behavior Tree Generation from Human Instructions

Xinglin Chen[*] , **Yishuai Cai**[*] , **Yunxin Mao** , **Minglong Li**[†] ,
Wenjing Yang , **Weixia Xu** and **Ji Wang**

College of Computer Science and Technology, National University of Defense Technology
{chenxinglin, caiyishuai, maoyunxin, liminglong10, wenjing.yang, xuweixia, wj}@nudt.edu.cn

Abstract

Robots executing tasks following human instructions in domestic or industrial environments essentially require both adaptability and reliability. Behavior Tree (BT) emerges as an appropriate control architecture for these scenarios due to its modularity and reactivity. Existing BT generation methods, however, either do not involve interpreting natural language or cannot theoretically guarantee the BTs' success. This paper proposes a two-stage framework for BT generation, which first employs large language models (LLMs) to interpret goals from high-level instructions, then constructs an efficient goal-specific BT through the Optimal Behavior Tree Expansion Algorithm (OBTEA). We represent goals as well-formed formulas in first-order logic, effectively bridging intent understanding and optimal behavior planning. Experiments in the service robot validate the proficiency of LLMs in producing grammatically correct and accurately interpreted goals, demonstrate OBTEA's superiority over the baseline BT Expansion algorithm in various metrics, and finally confirm the practical deployability of our framework. The project website is https://dids-ei.github.io/Project/LLM-OBTEA/.

1 Introduction

Issuing instructions using natural language is a convenient and user-friendly way to control robots in everyday life scenarios, such as home, restaurants, and factories. However, enabling robots to autonomously follow these instructions remains a challenge in the field of robotics and AI. On the one hand, due to the openness of natural language, the robot must understand the user's intents, clarify the goals of tasks, and acquire sufficient high-level skills to accomplish these tasks. On the other hand, the robot must automatically generate reliable plans and execute them safely and robustly.

Behavior Trees (BTs) employ a logical tree structure to control various high-level robot behaviors, which offer many advantages, including modularity, interpretability, reusability, reactivity, and robustness. Due to these characteristics, BTs have become a popular robot control architecture in recent years. Moreover, approaches to automatically generate BTs
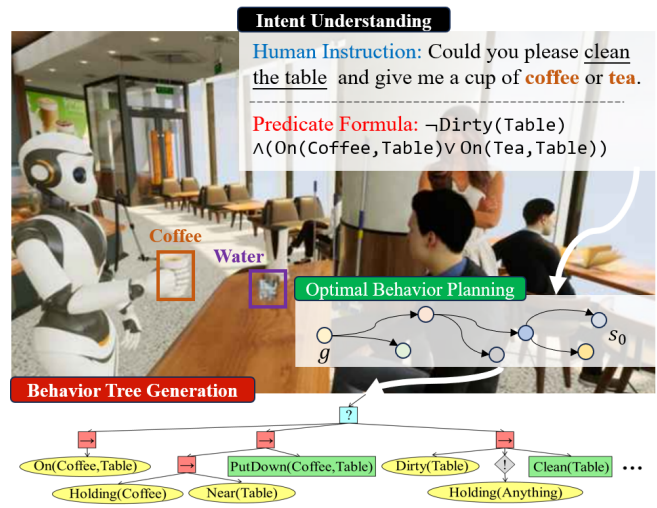

Figure 1: In our framework, human instructions are first understood and interpreted into a goal using LLMs. Then, OBTEA is employed to generate an optimal BT to achieve the goal. Goals are represented as well-formed formulas in first-order logic.

have also been widely studied, including evolutionary computing [Colledanchise *et al.*, 2019], reinforcement learning [Banerjee, 2018], and BT synthesis [Li *et al.*, 2021]. BT Expansion [Cai *et al.*, 2021] is a sound and complete algorithm for constructing BTs through behavior planning, theoretically ensuring the success of the BT. However, existing methods typically generate BTs from fitness functions, reward functions, or formalized goal conditions, but few studies have focused on generating BTs directly from natural language instructions.

Large Language Models (LLMs), trained on extensive corpora to generate text sequences based on input prompts, exhibit remarkable multi-task generalization capabilities. This ability has recently been used to generate plausible task plans, which can be represented as action sequences [Song *et al.*, 2023], codes [Singh *et al.*, 2022], and more. However, for the method of having LLMs directly generate BTs [Lykov and Tsetserukou, 2023], it not only necessitates gathering a large dataset of instruction-BT pairs for supervised fine-tuning, but also cannot guarantee theoretical correctness and optimality of the generated BTs.

This paper proposes a two-stage framework for efficient and reliable BT generation from human instructions. In the first stage, we utilize the LLM's capability in understanding intent to interpret a goal from the user's instructions. In the second stage, we design the Optimal Behavior Tree Expansion Algorithm (OBTEA) to construct a BT that theoretically can ensure the completion of the goal with minimal cost. The task goals, expressed as Well-Formed Formulas (WFFs) in first-order logic [Hodel, 2013], serve as a natural and ingenious bridge connecting these two stages. In a café scenario, for instance, there is a customer who says to the robot: *Could you please clean the table and give me a cup of coffee or tea?* The LLM generates a goal condition: $G = \neg \texttt{Dirty(Table)} \land (\texttt{On(Coffee,Table)} \lor \texttt{On(Tea,Table)})$. Then, OBTEA traverses the action and condition space to find actions that can be used to achieve the goal, such as `Clean(Table)` and `PutDown(Coffee,Table)`. These actions are then structured to generate the final optimal BT, as shown in Figure 1.

Our experiments demonstrate that the LLM (GPT 3.5), powered by few-shot demonstrations and reflective feedback, achieves high performance in both grammar and interpretation accuracy for intent understanding. The analysis of optimal behavior planning shows that the OBTEA produces BTs with lower costs and fewer condition node ticks compared to the baseline BT Expansion algorithm. Finally, the success of deployment in a café scenario highlights the promising future of our framework.

2 Preliminaries

Behavior Tree. A BT is a directed rooted tree structure in which the leaf nodes control the robot's perception and actions, while the internal nodes handle the logical structuring of these leaves [Colledanchise and Ögren, 2018]. At each time step, the BT ticks from its root. The tick, according to the environment state, creates a control flow within the tree, ultimately determining the action that the robot will execute. In this paper, we focus on five typical BT nodes:

- **Condition**: A leaf node that evaluates whether the observed state satisfies the specified condition, returning either `success` or `failure` based on this assessment.

- **Action**: A leaf node that controls the robot to perform an action, returning `success`, `failure`, `running` depending upon the outcome of execution.

- **Sequence**: An internal node that only returns `success` if all its children succeed. Otherwise, it ticks its children from left to right, and the first child to return `failure` or `running` will determine its return status. It is often visualized as a → inside a box.

- **Fallback**: An internal node with logic opposite to the sequence node. It returns `failure` only if all of its children fail. If not, the first occurrence of `success` or `running` during ticking becomes its return status. It is often visualized as a ? in a box.

- **Not-Decorator**: An internal node with a single child. It inverts the return statuses `success` and `failure` of its child. It is often visualized as a ! in a diamond.

Behavior Planning. Based on [Cai *et al.*, 2021] and [Colledanchise and Ögren, 2018], we describe the behavior planning problem as a tuple: $< \mathcal{S}, \mathcal{A}, P, s_0, g >$, where $\mathcal{S}$ is the finite set of environment states, $\mathcal{A}$ is the finite set of action, P defines the state transition rules, s_0 is the initial state, g is the goal condition. Both the state $s \in \mathcal{S}$ and the condition c are represented by a set of literals, similar to STRIPS-style planning [Fikes and Nilsson, 1971]. If $c \subseteq s$, it is said that the condition c holds in the state s. The state transition affected by the action $a \in \mathcal{A}$ can be defined as a triplet $P(a) = < pre(a), add(a), del(a) >$, comprising the precondition, add effects, and delete effects of the action. An action a can only be performed if $pre(a)$ holds in the current state s. After the execution of the action, the subsequent state s' will be:

$$s' = s \cup add(a) \setminus del(a) \tag{1}$$

In well-designed BT systems, for each literal or its negation, there is a condition node to verify its truthfulness, and for each action, there is a corresponding action node for implementation. Given the initial state s_0 and the goal condition g, the objective of behavior planning is to construct a BT $\mathcal{T}$ capable of transitioning s_0 to g in a finite time.

Problem Formulation. In this paper, we extend the problem to optimal behavior planning, as the user's instructions often imply that the robot should complete the ordered task at minimal cost. Considering a well-defined cost function D, where $\forall a \in \mathcal{A}, D(a) \geq 0$, for each BT $\mathcal{T}$ produced by behavior planning, if the actions required to transition from s_0 to g are $a_1, a_2, ..., a_n$, then the cost of the BT is calculated as $D(\mathcal{T}) = \sum_{i=1}^{n} D(a_i)$. Our aim is to identify the optimal BT, $\mathcal{T}_*$, which can achieve the goal condition g, as interpreted from human instructions, at the minimum cost.

3 Methodology

In this section, we first demonstrate how to represent goals as WFFs in first-order logic. Then, we dive into our two-stage framework for generating BT, where the input instruction is interpreted into the goal condition, and the OBTEA is applied to generate an optimal BT that guarantees to achieve the goal at the minimal cost. Our framework is illustrated in Figure 2.

3.1 Representing Goals as Well-Formed Formulas

To describe the environment in first-order logic, we introduce a triplet $\langle \mathcal{O}, \mathcal{P}_c, \mathcal{P}_a \rangle$, where $\mathcal{O}$ is the set of available objects in the environment, $\mathcal{P}_c$ is the set of condition predicates, and $\mathcal{P}_a$ is the set of action predicates. For a predicate $p \in \mathcal{P}_c \cup \mathcal{P}_a$, its domain may be a subset of the Cartesian product of $\mathcal{O}$. In this formulation, a literal can be denoted as $p_c(o_1, ..., o_k)$ while an action can be denoted as $p_a(o_1, ..., o_l)$. Literals can be further formed into WFFs using three fundamental logical connectors: $\land$ (conjunction), $\lor$ (disjunction), and $\neg$ (negation). The three fundamental connectors effectively cover a wide range of user intentions, and more intricate connectors may be integrated in future work. This high-level formulation is advantageous for LLM to understand and interpret, as well as for the design and implementation of modular and reusable behavior nodes.
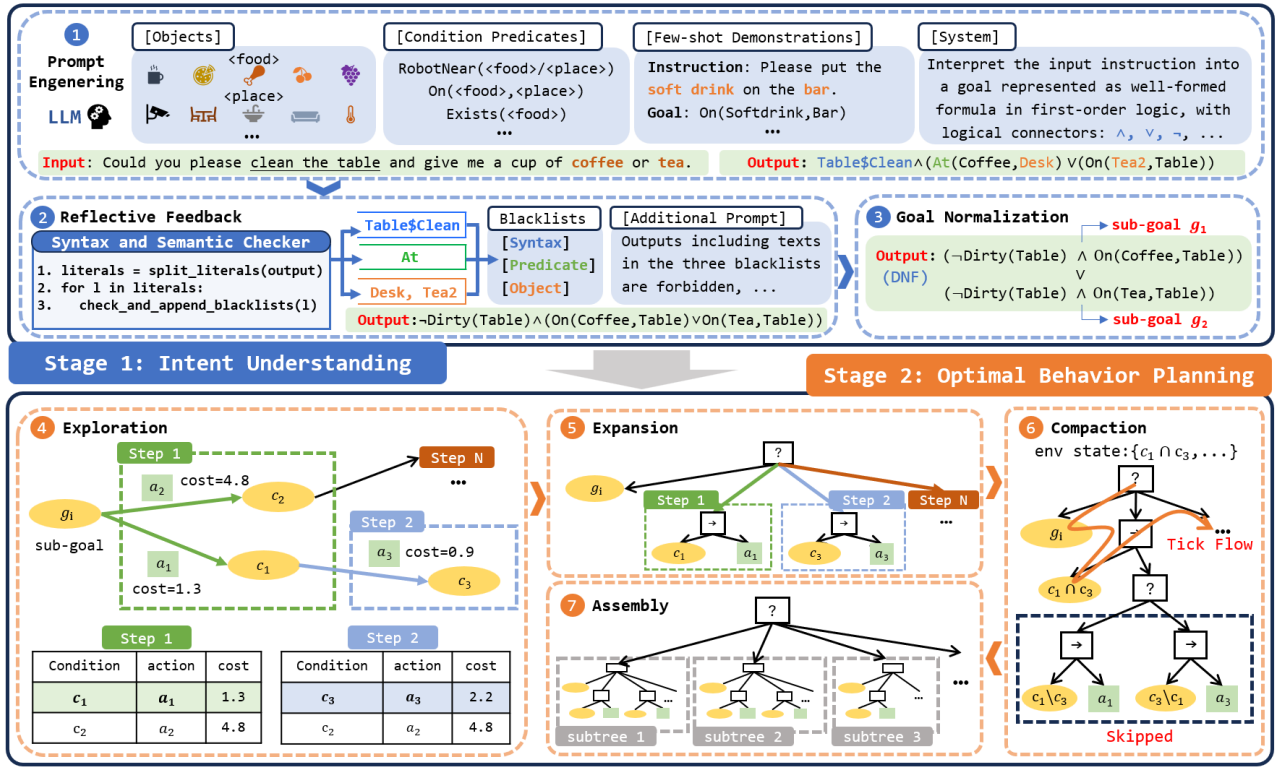
Figure 2: The two-stage framework. In Stage 1, the input instruction is transformed into a logically expressed goal by the LLM with prompt engineering and reflective feedback. The goal is then normalized as DNF and devided into sub-goals. In Stage 2, one subtree is generated for each sub-goal through exploration, expansion, and compaction. These subtrees are eventually assembled to create the final optimal BT.

3.2 Intent Understanding

Given the relatively interpretable nature of first-order logic grammar in everyday service scenarios, we believe that pre-trained LLMs have the potential to interpret the instruction without supervised fine-tuning on specialized datasets. We achieve this process through the following three methods.

Prompt Engineering. We deliberately design LLM prompts to provide context for goal generation, similar to [Singh *et al.*, 2022; Ahn *et al.*, 2022]. Our prompts consist of four components: (1) Objects. The set of available objects. To better define the domains of predicates, we categorize objects into several classes, such as <food> and <place>. (2) Condition Predicates. The set of condition predicates. The domain of a predicate is indicated by category, such as RobotNear(<place>) and On(<food>,<place>). (3) Few-shot Demonstrations. A few instruction-goal pairs as examples to teach the LLM on goal interpretation. The number of demonstrations does not need to be large, but it is beneficial to showcase as much diversity in both grammar and user's intention as possible. (4) System. An explanatory text provides context and the prompt for instructing the LLM to produce a goal in the correct format. We found that few-shot demonstrations have a significant impact on the LLM's ability to interpret goals, as language models have been shown to be few-shot learners [Brown *et al.*, 2020]. We conducted ablation experiments in Section 4.2 to analyze its impact.

Reflective Feedback. We introduce automatic reflective feedback to further enhance grammar accuracy, drawing inspiration from [Liu *et al.*, 2023b]. When the LLM produces an output, the syntax checker evaluates whether the output forms a WFF, while the semantic checker further verifies the validity of predicates and objects. If errors are detected, we re-input the instruction to the LLM with additional prompt to correct these errors. While we initially provide whitelists of objects and condition predicates in the original prompt, we augment the prompt with blacklists and additional context to instruct the LLM not to include the blacklisted texts in the output again. If errors persist, we continue to apply reflective feedback, gradually expanding the blacklists, until all grammar errors are corrected, or the maximum retry limit is reached.

Goal Normalization. A grammatically correct goal will ultimately undergo normalization into Disjunctive Normal Form (DNF). This normalization offers two significant advantages. Firstly, it effectively eliminates ambiguity from WFFs, which is crucial when assessing the equivalence of two goals, particularly in evaluating the semantic correctness of the model's outputs on the evaluation dataset. Secondly, each conjunction within DNF can be regarded as a sub-goal. When the robot accomplishes any one of these sub-goals, it is considered a success, as these sub-goals are represented as disjunctive relations. This enables us to design a modular algorithm for optimal behavior planning.

Algorithm 1 OBTEA

Input: Goal G, init state s_0, action set $\mathcal{A}$, transition rules P
Output: The optimal BT $\mathcal{T}_*$

1: $\mathcal{G}_{sub} \leftarrow ParseSubGoals(G)$ ▷ the set of sub goals
2: **for** $g_i \in \mathcal{G}_{sub}$ **do**
3: $\quad \mathcal{T} \leftarrow Fallback(g_i)$ ▷ init sub-BT
4: $\quad D(g_i) \leftarrow 0$ ▷ for $\forall c, c \neq g_i, D(c) \leftarrow +\infty$
5: $\quad \mathcal{C}_U \leftarrow \{g_i\}$ ▷ explored but unexpanded conditions
6: $\quad \mathcal{C}_E \leftarrow \emptyset$ ▷ expanded conditions
7: $\quad$ **while** $\mathcal{C}_U \neq \emptyset$ **do**
8: $\qquad c \leftarrow \arg\min_{c \in \mathcal{C}_U}(D(c))$ ▷ explore and expand c
9: $\qquad$ **for each** $a \in A$ **do**
10: $\qquad\quad$ **if** $(c \cap (pre(a) \cup add(a) \setminus del(a)) \neq \emptyset)$ and $(c \setminus del(a) = c)$ **then**
11: $\qquad\qquad c_a \leftarrow pre(a) \cup c \setminus add(a)$ ▷ **Exploration**
12: $\qquad\qquad$ **if** $c_a \notin \mathcal{C}_E$ and $D(c) + D(a) < D(c_a)$ **then**
13: $\qquad\qquad\quad D(c_a) \leftarrow D(c) + D(a)$ ▷ update $D(c_a)$
14: $\qquad\qquad\quad \mathcal{C}_U \leftarrow \mathcal{C}_U \cup \{c_a\}$ ▷ add c_a to $\mathcal{C}_U$
15: $\qquad\qquad\quad \mathcal{M}(c_a) \leftarrow Sequence(c_a, a)$
16: $\qquad \mathcal{C}_U \leftarrow \mathcal{C}_U \setminus \{c\}$ ▷ remove c from $\mathcal{C}_U$
17: $\qquad$ **if** $c \neq g_i$ **then**
18: $\qquad\quad \mathcal{T} \leftarrow Fallback(\mathcal{T}, \mathcal{M}(c))$ ▷ **Expansion**
19: $\qquad$ **if** $c \subseteq s_0$ **then**
20: $\qquad\quad D(\mathcal{T}) \leftarrow D(c)$ ▷ total cost of $\mathcal{T}$
21: $\qquad\quad$ **break** ▷ $\mathcal{T}$ is the optimal BT for sub-goal g_i
22: $\qquad \mathcal{C}_E \leftarrow \mathcal{C}_E \cup \{c\}$ ▷ add c to $\mathcal{C}_E$
23: $\quad \mathcal{T}_i \leftarrow Compact(\mathcal{T})$ ▷ **Compaction**
24: $\mathcal{T}_1', \mathcal{T}_2', ..., \mathcal{T}_{|\mathcal{G}_{sub}|}' = \underset{key=D(\mathcal{T})}{SortAscending}(\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_{|\mathcal{G}_{sub}|})$
25: $\mathcal{T}_* \leftarrow Fallback(\mathcal{T}_1', \mathcal{T}_2', ..., \mathcal{T}_{|\mathcal{G}_{sub}|}')$ ▷ **Assembly**
26: **return** $\mathcal{T}_*$

3.3 Optimal Behavior Planning

We introduce the OBTEA for optimal behavior planning based on the goal G represented in DNF (Fig. 2 and Alg. 1). At the beginning of OBTEA, we parse the goal G to identify sub-goals separated by $\vee$. Each sub-goal is then further divided by $\wedge$ to form a set of literals, each consisting of a goal condition g_i. This process results in a set of goal conditions that constitute the sub-goal set $\mathcal{G}_{sub}$ (line 1). For example, if the goal in DNF is $G = (l_1 \wedge l_2) \vee (l_1 \wedge l_3)$, this translates to two sub-goals: $l_1 \wedge l_2$ and $l_1 \wedge l_3$. Consequently, the sub-goal condition set becomes $\mathcal{G}_{sub} = \{g_1, g_2\}$, where $g_1 = \{l_1, l_2\}$ and $g_2 = \{l_1, l_3\}$. Note that we facilitate more flexible human instructions by allowing negative literals $\neg l$, which are often banned in action planners for performance reasons. Users must weigh the trade-off between interpretability and computational complexity in practical applications.

For each sub-goal $g_i \in \mathcal{G}_{sub}$, we construct an optimal subtree (line 2). Initially, each subtree is rooted in a fallback node, with its child node being the sub-goal condition g_i (line 3). To track the shortest distance, we maintain a cost function D, which is initialized with $D(c_g) = 0$ and $D(c) = +\infty, \forall c \neq g_i$ (line 4). Furthermore, we maintain two sets: $\mathcal{C}_U$ and $\mathcal{C}_E$. $\mathcal{C}_U$ encompasses explored but unexpanded conditions, initialized with $\{g_i\}$ (line 5). $\mathcal{C}_E$ includes conditions that have been expanded, initially empty (lines 6).

Exploration. We continuously pop the condition $c \in \mathcal{C}_U$ with the minimum cost (line 8). This process aims to explore the condition space following the shortest path from g_i. For each popped condition c, we explore its neighboring conditions by iterating through all available actions (line 9) and identifying those that can reach the condition c after execution. The neighboring conditions are generated similarly to the process in the BT Expansion [Cai *et al.*, 2021] (lines 10-11), and can be denoted as: $c_a \leftarrow pre(a) \cup c \setminus add(a)$. Then we disregard c_a in the following two cases (line 12): (1) $c_a \in \mathcal{C}_E$, indicating that it has already been expanded and its shortest path is found. (2) $D(c) + D(a) \geq D(c_a)$, showing that c_a has been explored and the path cost found is equal or lower. In both cases, further exploration or updates of c_a are unnecessary. If neither case applies, we explore c_a by updating $D(c_a)$, adding c_a to $\mathcal{C}_U$, and recording the subtree $Sequence(c_a, a)$ leading to c_a (line 13-15). After exploring all the neighbors of c, it is removed from $\mathcal{C}_U$ (line 16).

Expansion. After exploring c, we integrate it into the current BT $\mathcal{T}$. This involves adding the subtree $\mathcal{M}(c)$ as a child node of the root fallback node (line 18). The algorithm continues to explore the condition space until it expands a condition c that holds in the initial state s_0 (line 19). At this point, we record the total cost of the subtree for g_i as $D(\mathcal{T}) = D(c)$ (line 20), and the exploration-expansion loop is immediately terminated (line 21). If s_0 is not reached, we add c to the expanded set $\mathcal{C}_E$ (line 22). It is important to note that this method of expansion might include conditions that are not part of the shortest path. However, this does not affect the optimality since these extra conditions will not be visited along the shortest path. This approach enhances the BT's reactivity and robustness when the state deviates from the original path, ensuring that the path starting from each expanded condition is optimal.

Compaction. For an expanded optimal subtree, we propose a simple but efficient method to compact its structure and improve runtime efficiency. During compaction, we recursively extract common literals from two adjacent conditions to create smaller conditions for checking. Consider a BT produced through the exploration and expansion process. If two adjacent subtrees have conditions c_1 and c_3, respectively (as illustrated in Fig. 2), both conditions need to be checked during runtime. After compaction, if $c_1 \cap c_3 \neq \emptyset$, a new condition $c' = c_1 \cap c_3$ will be created and checked first during ticking. If it does not hold, the conditions $c_1 \setminus c_3$ and $c_3 \setminus c_1$ will be skipped, thereby saving time. The compacted subtrees can recursively join the compaction with its neighbors due to the modularity of BT. We evaluated the impact of the maximum recursion depth during compaction on condition check count in Section 4.3.

Assembly. Once all subtrees have been generated, we can assemble the final optimal BT. We start by sorting all the subtrees in ascending order according to their total costs (line 24). This ensures that the BT will prioritize the feasible paths with the lowest cost. Then, we construct the final BT $\mathcal{T}_*$ by adding the sorted subtrees under a fallback node (line 25). After this, the algorithm ends and returns $\mathcal{T}_*$ as the optimal BT to achieve the final goal G at minimal cost.

Difficulty	Data	Examples	Goals	Demonstrations	GA-NF(%)	GA-1F(%)	GA-5F(%)	IA(%)
Easy	30	Please come to the bar.	`RobotNear(Bar)`	Zero-Shot	60.0	74.7	91.3	56.7
				Few-shot-1	87.3	95.3	100.0	85.3
				Few-shot-5	92.0	96.7	**100.0**	**90.7**
Medium	30	Ensure there's enough water and keep the hall light on.	`Exist(Water) ∧` `Active(HallLight)`	Zero-Shot	50.0	66.0	88.0	57.3
				Few-shot-1	77.3	91.3	98.7	62.0
				Few-shot-5	92.7	94.7	**99.3**	**82.0**

Hard	40	I'm at table three, please bring yogurt and keep the AC warm. If there is no yogurt, coffee is fine.	`(On(Yogurt,Table3) ∧`
			`(On(Coffee,Table3)) ∨`
			`¬ Low(ACTemperature)`
				Zero-Shot	48.0	62.5	84.0	42.0

				Few-shot-1	76.5	87.5	93.0	51.0
				Few-shot-5	80.5	88.0	**96.0**	**65.0**

Table 1: Performance analysis of GPT 3.5 on intent understanting in the café sencario.

Evaluation

The BT generated by OBTEA is theoretically guaranteed to be finite-time successful and optimal, assuming that all sub-goals can be reached from the initial state s_0 within the BT system. Similar to BT Expansion [Cai *et al.*, 2021], the time complexity of OBTEA is also $O(|\mathcal{A}||\mathcal{S}| \log(|\mathcal{S}|))$ in the worst case, which is polynomial in terms of the system size. Proofs and a detailed discussion are in the appendix.

Case	Total Costs		Condition Node Ticks	
	Baseline	OBTEA	Baseline	OBTEA
Easy	123.7	**67.4**	65.4	**14.2**
Medium	171.1	**92.9**	271.1	**105.0**
Hard	204.8	**111.7**	1437.4	**809.7**

Table 2: Performance comparison on optimal behavior planning in the café scenario.

4 Experiments

4.1 Settings

We conducted experiments and deployed our framework in a hospitality industry scenario, where the robot plays the role of a waiter in a café. It is responsible for serving customers and performing various tasks following human instructions. In this scenario, the design of object and action sets is similar to the Virtual Home [Puig *et al.*, 2018] and Alfred [Shridhar *et al.*, 2020] benchmarks, but it is tailored to better suit the specific requirements of a café. The scenario includes a total of 80 available objects, 8 condition predicates: $\mathcal{P}_c = $ {`RobotNear,Holding,On,Closed,Exists,Dirty,Activate,Low`} and 6 action predicates $\mathcal{P}_a = $ {`Make, MoveTo,PickUp,PutDown,Clean,Turn`}. Different predicates have different domains and implementations on behavior nodes, resulting in a total of 1318 extendable literals and 1269 extendable actions.

4.2 Analysis for Intent Understanding

To evaluate the intent understanding capabilities of LLM for goal interpretation, we created an evaluation dataset comprising 100 instructions and their corresponding expected goals, categorized into three difficulty levels. As exampled in Table 1, the Easy set consists of goals formed by a single positive literal. In the Medium set, goals consist of two literals connected by ∨ or ∧. The Hard set involves three literals that incorporate ∨, ∧, and ¬. The performance of the LLM in the interpretation of goals is evaluated primarily based on two metrics: grammatical accuracy (GA) and interpretation accuracy (IA). GA is the fraction of output goals that are free from grammatical errors. IA is the fraction of goals that are equivalent to the ground truth.

Our methods are model-agnostic, allowing for application to any LLM. We conducted the experiments with GPT-3.5, specifically using the gpt-3.5-turbo model (2024.06). Table 1

presents the performance of the model in three levels of difficulty, with each result being the average of 5 runs. We studied the impact of two factors on our method: the number of demonstrations and the maximum retry limits. Our standard setting involves five demonstrations with a limit of five retries for reflective feedback. At the easy level, we find the LLM performance is surprisingly high, with GA and IA reaching 100%, 99.3% and 96%, respectively. At medium and hard levels, the decrease in IA is faster than in GA. This suggests that the grammar of first-order logic is easier for LLMs to learn, but the translation from natural language is more challenging. Overall, both GA and IA are relatively high even at the hard level, demonstrating the LLM's potential for interpreting WFFs even without any supervised fine-tuning.

Ablating Few-shot Demonstration. To conduct the ablation analysis on the number of demonstrations, we reduced the number of demonstrations while keeping the objects, predicates, and system explanations constant. It is observed that when no demonstrations are provided (zero-shot), both GA and IA significantly decrease across three levels of difficulty. Furthermore, IA is particularly lower than GA in the zero-shot setting, which again confirms the challenge for LLM to learn intent understanding rather than learning logical grammar only in explanatory prompts. Without any demonstrations, the pre-trained LLM struggles to provide WFF-represented goals that align with the user's intent. However, there is a notable improvement in performance after adding just one demonstration (one-shot). This phenomenon indicates that one example can be more effective than a lengthy explanation. The results underscore the importance of including examples in prompts and also highlight the LLM's few-shot learning ability in interpreting goals.

Ablating Flextive Feedback. We then performed the ablation analysis on the maximum retry times with reflective feedback. The results show that without feedback (GA-NF), the

Case	Scenario Configuration				Problem			Total Costs		Condition Node Ticks		
	$\lvert\mathcal{O}\rvert$	$\lvert\mathcal{P}_c\rvert$	$\lvert\mathcal{P}_a\rvert$	MAC	Literals	States	Actions	Baseline	OBTEA	Baseline	OBTEA	OBTEA-NC
0	100	10	10	0	995.4	475.5	509.5	71.3	**58.5**	31.7	**23.8**	33.1
1	100	10	50	0	1000.0	4046.5	2495.9	52.0	**26.8**	25.7	**22.7**	33.8
2	500	50	50	0	25000.0	22468.6	12508.2	164.2	**137.0**	950.4	**632.3**	1779.6
3	100	10	10	5	995.1	466.1	527.3	71.7	**26.1**	54.0	**28.5**	54.7
4	100	30	10	5	2984.4	501.0	530.4	170.9	**81.1**	597.1	**358.2**	1005.1
5	100	50	10	5	4974.6	497.7	524.4	241.4	**124.8**	1581.2	**1210.5**	3578.8
6	100	50	30	5	5000.0	2496.0	1527.4	189.4	**97.7**	1391.9	**1147.9**	3111.3
7	100	50	50	5	5000.0	4501.6	2531.5	169.4	**85.1**	1584.4	**971.8**	2848.0
8	300	50	50	5	15000.0	13550.7	7560.9	164.2	**82.5**	1302.4	**892.8**	2577.2
9	500	50	50	5	25000.0	22427.9	12514.2	165.9	**82.7**	1436.6	**909.8**	2581.2

Table 3: Performance comparison on behavior planning in the computational test sets. Each values are averaged over 1000 runs.

performance is markedly decreased, especially in the absence of demonstrations. Similarly to the number of demonstrations, GA improved when the LLM is allowed to retry only once (GA-1F), and an increase in the number of feedback times continuously improved grammar correctness. These results confirm the effectiveness of reflective feedback.

4.3 Analysis for Optimal Behavior Planning

Once the LLM produces the correct goal, OBTEA is guaranteed to generate an optimal BT to achieve that goal. We evaluated the BT generation performance for 100 ground-truth goals in the café scenario, where the action cost is set based on human experience. We focus on two key metrics: total costs of the BT and the condition node ticks during the BT's execution. Table 2 shows the performance comparison between OBTEA and the baseline BT Expansion algorithm [Cai *et al.*, 2021]. The results demonstrate that our OBTEA generates BTs with lower costs and fewer condition node ticks than the baseline, and thereby has higher execution efficiency.

Random Test Sets. In addition to the dataset in the café scenario, we also designed random test sets to verify the generality of OBTEA. Inspired by [Cai *et al.*, 2021], we randomly generate various optimal behavior planning tasks by creating action sequence paths from the initial state to the goal and then randomly adding other actions. In contrast to previous work, our datasets are represented using first-order logic. The problem scale is mainly determined by the number of objects ($\mathcal{O}$), condition predicates ($\mathcal{P}_c$), and action predicates ($\mathcal{P}_a$). To increase the number of valid paths to the goal, we can randomly copy the actions in the original path several times, but assign different costs, denoted as maximum action copies (MAC). Further details can be found in the appendix.

Table 3 shows the results averaged over 1000 runs. The results again emphasize OBTEA's proficiency in achieving lower action costs and condition node ticks across all cases with varying complexity. In particular, in the most complex case 9, OBTEA significantly reduces costs by more than 50%, and condition node ticks decrease by approximately 30%. When there are more available paths to reach the goal, the costs of BTs found by OBTEA are smaller than the baseline. Note that while OBTEA has achieved improvements, its additional time overhead is also acceptable. The test results and discussions regarding planning time are in the appendix.
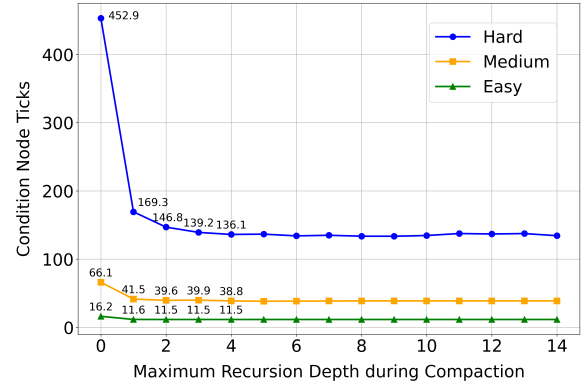

Figure 3: The impact of maximum recursion depth on condition node ticks during compaction in the café scenario.

Ablating Compaction. We then performed an ablation analysis on the impact of the maximum recursion depth during compaction. Table 3 shows that when we remove compaction (OBTEA-NC), the condition node ticks increase significantly in all cases. Experiments in café scenarios (Figure 3) show that condition node ticks decrease rapidly as the recursion depth increases when the recursion depth is small and then become rather constant when we continuously increase the depth. This means that the recursion depth does not need to be set too large to achieve optimal performance in the café scenario.

4.4 Deployment

We implemented our framework in a digitally twinned café environment constructed using the simulator used in MO-VLN [Liang *et al.*, 2023], as shown in Figure 4. In the simulator, a humanoid robot acts as a waiter and has 21 active joints for precise movement control. We implement 8 condition nodes and 6 action nodes, corresponding to conditions and action predicates. To gather object information, the robot initially performs visual segmentation and detection to determine object poses from multiple perspectives. For efficient condition checks, the robot actively explores and continuously updates a semantic topological map for task execution queries. By integrating these technologies, we ultimately achieve an adaptable, comprehensive, and reliable café waiter
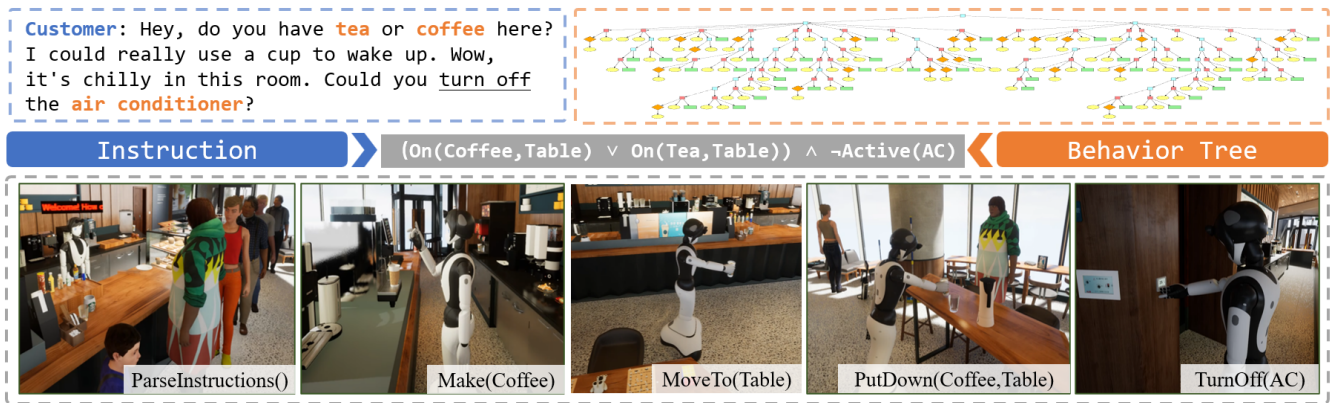
Figure 4: An example of the robot executing tasks follows the customer's instruction. After receiving the instruction at the bar, the robot generates an optimal BT through intent understanding and optimal behavior planning. The robot then autonomously completes the goal under the BT's control, which involves making the coffee, delivering it to the customer, and turning off the air conditioner.

robot system capable of visual-language navigation, manipulation, and handling complex combination tasks. Our work was presented and won the First Prize in a competition at ChinaSoft 2023. The successful deployment demonstrates the utility and practicality of our framework, representing a promising behavior tree generation framework in embodied intelligence scenarios.

5 Related Work

BT Generation. The study of automatic BT generation is a significant area of research in robotics and AI. Various methods have been explored for this purpose. Heuristic search methods, such as grammatical evolution [Neupane and Goodrich, 2019], genetic programming [Colledanchise *et al.*, 2019; Lim *et al.*, 2010], and Monte Carlo DAG Search [Scheide *et al.*, 2021], have been widely investigated. In addition to heuristic approaches, machine learning techniques have been applied to generate BTs. In particular, methods such as reinforcement learning [Banerjee, 2018; Pereira and Engel, 2015] and imitation learning [French *et al.*, 2019] have gained attention. There are also research efforts that employ formal synthesis methods to generate BTs, such as Linear Temporal Logic (LTL) [Li *et al.*, 2021] and its variant approaches [Tadewos *et al.*, 2022; Neupane and Goodrich, 2023]. However, it is worth noting that this approach can be resource intensive in terms of modeling the environment and the computational resources required to solve the problem.

LLMs for Task Planning. LLMs have gradually gained widespread application in the field of robot task planning recently. Some research focuses on directly controlling the low-level actions of robots using multi-modal language models [Driess *et al.*, 2023], while others use large language models to generate high-level behavior planning for robots. Generating BTs based on human instructions is an area that has seen limited exploration. One straightforward approach is to train a large language model to generate behavior tree structures represented in XML format [Lykov and Tsetserukou, 2023; Lykov *et al.*, 2023]. This approach is similar to the methods where LLMs directly generate action sequences [Song

et al., 2023; Liu *et al.*, 2023a; Ahn *et al.*, 2022; Chen *et al.*, 2023], or codes [Liang *et al.*, 2022; Singh *et al.*, 2022; Zeng *et al.*, 2022]. However, these approaches may not fully leverage the modularity and computability of BTs for planning, potentially resulting in a loss of the reliability and interpretability advantages that BTs offer.

Our approach combines the ability of LLMs to understand intent with first-order logical reasoning for correctness assurance, ultimately achieving BT generation based on human instructions. This approach aligns with the concepts of neurosymbolic approaches [Chaudhuri *et al.*, 2021] and symbolic planning [Liu *et al.*, 2023b]. In the future, it is possible to enhance the generalization of robot planning in new scenarios by combining techniques like rule discovery [Zhu *et al.*, 2023]. Hence, our framework is considered a promising approach for creating interpretable and reliable embodied intelligent agents.

6 Conclusion

This paper proposes a two-stage framework for BT generation, leveraging the strengths of LLMs for intent understanding and introducing the OBTEA for optimal behavior planning. The goal is represented as a well-formed formula in first-order logic, serving as a natural and innovative link between these two stages. Our framework integrates intent understanding and optimal behavior planning, which shows promise in creating adaptable and reliable generative embodied agents. In addition, we deployed our framework in a café scenario, showcasing its significant potential.

In the future, many tasks can be undertaken. For example, we can test and improve the effectiveness of goal interpretation on more benchmarks such as the RoboCup@Home competition. We can enhance the OBTEA's planning efficiency using heuristics like modern planners. Last but not least, we can integrate rule discovery algorithms to automatically learn environment transition rules from experience, thereby enhancing robots' capabilities for rapidly adapting and planning BTs in unseen scenarios.

Acknowledgments

Contribution Statement

The contributions of Xinglin Chen and Yishuai Cai to this paper were equal. Corresponding author: Minglong Li.

References

[Ahn *et al.*, 2022] 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.

[Banerjee, 2018] Bikramjit Banerjee. Autonomous Acquisition of Behavior Trees for Robot Control. pages 3460–3467, 2018.

[Brown *et al.*, 2020] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. 2020.

[Cai *et al.*, 2021] Zhongxuan Cai, Minglong Li, Wanrong Huang, and Wenjing Yang. BT Expansion: A Sound and Complete Algorithm for Behavior Planning of Intelligent Robots with Behavior Trees. pages 6058–6065. AAAI Press, 2021.

[Chaudhuri *et al.*, 2021] Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, Yisong Yue, et al. Neurosymbolic programming. *Foundations and Trends® in Programming Languages*, 7(3):158–243, 2021.

[Chen *et al.*, 2023] Yaran Chen, Wenbo Cui, Yuanwen Chen, Mining Tan, Xinyao Zhang, Dongbin Zhao, and He Wang. RoboGPT: An intelligent agent of making embodied long-term decisions for daily instruction tasks. *arXiv preprint arXiv:2311.15649*, 2023.

[Colledanchise and Ögren, 2018] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018.

[Colledanchise *et al.*, 2019] Michele Colledanchise, Ramviyas Parasuraman, and Petter Ögren. Learning of Behavior Trees for Autonomous Agents. *IEEE Transactions on Games*, 11(2):183–189, 2019.

[Driess *et al.*, 2023] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378*, 2023.

[Fikes and Nilsson, 1971] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. 2(3-4):189–208, December 1971.

[French *et al.*, 2019] Kevin French, Shiyu Wu, Tianyang Pan, Zheming Zhou, and Odest Chadwicke Jenkins. Learning behavior trees from demonstration. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 7791–7797. IEEE, 2019.

[Hodel, 2013] Richard E Hodel. *An Introduction to Mathematical Logic*. Courier Corporation, 2013.

[Li *et al.*, 2021] Shen Li, Daehyung Park, Yoonchang Sung, Julie A Shah, and Nicholas Roy. Reactive task and motion planning under temporal logic specifications. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 12618–12624. IEEE, 2021.

[Liang *et al.*, 2022] 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 *arXiv Preprint arXiv:2209.07753*, 2022.

[Liang *et al.*, 2023] Xiwen Liang, Liang Ma, Shanshan Guo, Jianhua Han, Hang Xu, Shikui Ma, and Xiaodan Liang. MO-VLN: A Multi-Task Benchmark for Open-set Zero-Shot Vision-and-Language Navigation. *arXiv*, 2023.

[Lim *et al.*, 2010] Chong-U Lim, Robin Baumgarten, and Simon Colton. Evolving behaviour trees for the commercial game DEFCON. In *Applications of Evolutionary Computation: EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*, pages 100–110. Springer, 2010.

[Liu *et al.*, 2023a] 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*, 2023.

[Liu *et al.*, 2023b] Chengwu Liu, Jianhao Shen, Huajian Xin, Zhengying Liu, Ye Yuan, Haiming Wang, Wei Ju, Chuanyang Zheng, Yichun Yin, Lin Li, et al. Fimo: A challenge formal dataset for automated theorem proving. *arXiv preprint arXiv:2309.04295*, 2023.

[Lykov and Tsetserukou, 2023] Artem Lykov and Dzmitry Tsetserukou. LLM-BRAIn: AI-driven Fast Generation of Robot Behaviour Tree based on Large Language Model. *arXiv preprint arXiv:2305.19352*, 2023.

[Lykov *et al.*, 2023] Artem Lykov, Maria Dronova, Nikolay Naglov, Mikhail Litvinov, Sergei Satsevich, Artem Bazhenov, Vladimir Berman, Aleksei Shcherbak, and Dzmitry Tsetserukou. LLM-MARS: Large Language Model for Behavior Tree Generation and NLP-enhanced

Dialogue in Multi-Agent Robot Systems. *arXiv preprint arXiv:2312.09348*, 2023.

[Neupane and Goodrich, 2019] Aadesh Neupane and Michael Goodrich. Learning Swarm Behaviors using Grammatical Evolution and Behavior Trees. pages 513–520. IJCAI Organization, 2019.

[Neupane and Goodrich, 2023] Aadesh Neupane and Michael A Goodrich. Designing Behavior Trees from Goal-Oriented LTLf Formulas. *arXiv preprint arXiv:2307.06399*, 2023.

[Pereira and Engel, 2015] Renato de Pontes Pereira and Paulo Martins Engel. A framework for constrained and adaptive behavior-based agents. *arXiv preprint arXiv:1506.02312*, 2015.

[Puig *et al.*, 2018] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018.

[Scheide *et al.*, 2021] Emily Scheide, Graeme Best, and Geoffrey A. Hollinger. Behavior Tree Learning for Robotic Task Planning through Monte Carlo DAG Search over a Formal Grammar. pages 4837–4843, Xi'an, China, May 2021. IEEE.

[Shridhar *et al.*, 2020] 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.

[Singh *et al.*, 2022] 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. September 2022.

[Song *et al.*, 2023] Chan Hee Song, Jiaman Wu, Clayton Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 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, 2023.

[Tadewos *et al.*, 2022] Tadewos G Tadewos, Abdullah Al Redwan Newaz, and Ali Karimoddini. Specification-guided behavior tree synthesis and execution for coordination of autonomous systems. *Expert Systems with Applications*, 201:117022, 2022.

[Zeng *et al.*, 2022] Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[Zhu *et al.*, 2023] Zhaocheng Zhu, Yuan Xue, Xinyun Chen, Denny Zhou, Jian Tang, Dale Schuurmans, and Hanjun Dai. Large Language Models can Learn Rules. *arXiv*, October 2023.