2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

LLM-BT: Performing Robotic Adaptive Tasks based on Large Language Models and Behavior Trees

Haotian Zhou[1†], Yunhan Lin[1†], Longwu Yan[1], Jihong Zhu[2] and Huasong Min[1*]

***Abstract*— Large Language Models (LLMs) have been widely utilized to perform complex robotic tasks. However, handling external disturbances during tasks is still an open challenge. This paper proposes a novel method to achieve robotic adaptive tasks based on LLMs and Behavior Trees (BTs). It utilizes ChatGPT to reason the descriptive steps of tasks. In order to enable ChatGPT to understand the environment, semantic maps are constructed by an object recognition algorithm. Then, we design a Parser module based on Bidirectional Encoder Representations from Transformers (BERT) to parse these steps into initial BTs. Subsequently, a BTs Update algorithm is proposed to expand the initial BTs dynamically to control robots to perform adaptive tasks. Different from other LLM-based methods for complex robotic tasks, our method outputs variable BTs that can add and execute new actions according to environmental changes, which is robust to external disturbances. Our method is validated with simulation in different practical scenarios.**

I. Introduction

Large Language Models (LLMs) [1] demonstrate powerful reasoning capabilities in robotics. By utilizing LLMs, robots can efficiently understand user intentions [2] and deduce the workflow of tasks [3]. However, the application of LLMs in complex robotic tasks faces challenges. One challenge is that the descriptive steps generated by LLMs are not executable without knowledge of operational skills.

Recently, there are many methods for solving the grounding problems of LLMs in robotics. PaLM-E [4] generates control sentences according to multi-modal data. RT-2 [5] directly infer instructions based on languages and images. ChatGPT for Robotics [6] needs the declaration of APIs for reasoning the actions of tasks. SayCan [7] selects most suitable actions according to environmental information. VoxPoser [8] converts the observation space into a 3D value maps for generating trajectories. These methods are able to achieve robotic autonomous tasks and handle some partial disturbances. For example, when objects drop from the manipulator of robots, they can re-pick them.

However, they cannot handle external disturbances that require re-planning. For example, when an obstacle O_b prevents the robot from placing object O_a at position P_a, they cannot put down O_a, then move O_b away, and finally pick O_a and place it at P_a. This is because these methods are unable to add new actions with a higher executing priority at runtime.

Behavior Trees (BTs) [9] were popular in video games [10] and are widely used in robotics due to their unique properties of modularity and reactivity [11]. BTs can react to environmental changes by frequent traversal (which calls tick) to activate actions [12]. The synthesis of BTs [13] allows new actions to be added with a higher executing priority in real time, which enables robots to deal with external disturbances. However, this kind of method needs to pre-define the goal of tasks.

In this paper, we utilize LLMs to construct initial BTs that include the goal of tasks and propose a BTs Update algorithm to expand the initial BTs. Fig. 1 shows the overview of our method which calls LLM-BT[1].

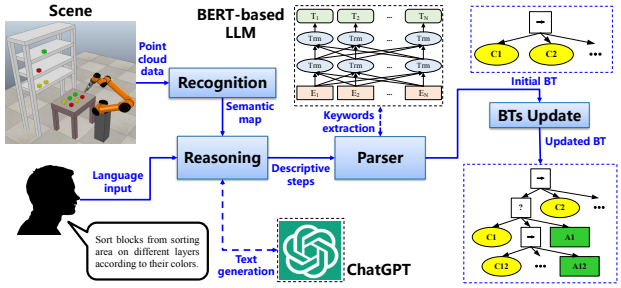

Fig. 1. Overview of LLM-BT.

The Recognition module obtains information of objects in real-time scene to construct semantic maps. In the Reasoning module, ChatGPT [14] is adopted to deduce the descriptive steps of tasks based on the user inputs and semantic maps. Next, the Parser module utilizes a Bidirectional Encoder Representations from Transformers (BERT)-based [15] LLM to extract keywords from the descriptive steps and then constructs initial BTs. Finally, a BTs Update algorithm is proposed to expand the initial BTs dynamically, which add and execute actions iteratively to satisfy the goal of tasks.

Compared to other LLM-based methods for complex robotic tasks, we have two advantages.

(1) Adaptability: LLM-BT outputs variable BTs that can add and execute new actions according to environmental changes, which is robust to external disturbances, as shown in Table I.

[1]https://github.com/henryhaotian/LLM-BT

[†]These authors contributed equally to this work. *Coresponding author: Huasong Min. This work is supported by the National Key R&D Program of China (grant No.: 2022YFB4700400), National Natural Science Foundation of China (grant No.: 62073249), Key R&D Program of Hubei Province (grant No.: 2023BBB011).

[1]Haotian Zhou, Yunhan Lin, Longwu Yan and Huasong Min are with Institute of Robotics and Intelligent Systems, Wuhan University of Science and Technology, Wuhan, China. zhtwust@163.com, yhlin@wust.edu.cn, longwuyan0621@163.com, mhuasong@wust.edu.cn.

[2]Jihong Zhu is with the School of Physics, Engineering and Technology, University of York, the UK. jihong.zhu@york.ac.uk.

(2) Modularity: LLM-BT consists of four modules, and each module can adopt advanced algorithms in the future.

TABLE I
COMPARISON WITH LLM-BASED METHODS FOR COMPLEX ROBOTIC TASKS.

Methods	Input	Output
LLM-BT(Ours)	language, images	variable BTs
PaLM-E [4]	languages, multi-modal data	control sentences
RT-2 [5]	languages, images	low-level instructions
ChatGPT for Robotics [6]	languages, declaration of APIs	actions linked to APIs
SayCan [7]	languages, images	selected actions
VoxPoser [8]	languages, images	motion trajectories

The main contributions of this paper are as follows:

(1) We propose a novel method to perform robotic adaptive tasks by constructing BTs automatically and expanding them dynamically. To the best of our knowledge, this is the first time that BTs have been generated through LLMs.

(2) We compare our method with advanced LLM-based methods and discuss the advantages and limitations of our method.

The rest of this paper is organized as follows. Section II is the related work. Section III introduces BTs briefly. Section IV illustrates the proposed method. Section V provides the experiments. Section VI is a discussion of our method and Section VII lists the conclusion and future work.

II. RELATED WORK

A. LLM-based methods for complex robotic tasks

PaLM-E [4] integrated an advanced LLM and a vision model for end-to-end training. It utilized text and multi-modal data (images, robot states, environmental information, etc.) instead of pure text as input, and outputted control sentences.

RT-1 [16] adopted imitation learning to acquire skills for robots. It first converted text instructions and images into tokens, and then used a Token Learner to compress these tokens in order to improve the model's inference speed. Finally, the compressed tokens were transmitted into a Transformer for training. Subsequently, based on RT-1, RT-2 [5] used a dataset of robot skills to fine-tune model, which significantly enhancing the capability of task generalization.

ChatGPT for Robotics [6] created an advanced function library and then linked it to API of actual robotic platform. So, the robot can parse user intention and convert it into high-level function calls.

SayCan [7] combined LLMs with Value Function. It used LLMs to output available high-level actions. Then, the value Function assigned scores to these actions based on environmental information. The scores represented the probability of these actions being executed respectively. Therefore, the action with a highest score would be executed.

VoxPoser [8] used LLMs and Vision-Language Models (VLMs) to convert the observation space into a 3D value maps, and then adopted mature motion planning algorithms to achieve adaptive tasks.

Differ from these methods, LLM-BT outputs variable BTs that can add new actions and assign their priorities based on environmental changes.

B. Synthesis of BTs

Excepts the method that synthesizes BTs based on *Blended Reactive Planning and Acting* [13], advanced AI algorithms are utilized for the acquisition of BTs in many approaches.

Scheper, et al [17] employed genetic algorithms to select, crossover, and mutate nodes or subtrees in BTs. However, a complete BT must be manually constructed at first for subsequent optimization. French, et al [18] learned BTs from demonstration. They used Classification and Regression Tree algorithm to generate decision trees from human demonstrations and then converted the decision trees into BTs. Banerjee [19] learned BTs by reinforcement learning, which designed a conversion rule to transform the trained Q-tables into BTs.

The BTs Update algorithm in our method is related to [13]. The difference is that we construct the initial BTs that include the goal of tasks by LLMs.

III. BACKGROUND:BEHAVIOR TREES

A BT consists of control flow nodes, execution nodes and a root node. Control flow nodes traverse their child nodes based on some logic. Execution nodes are able to check conditions or perform actions. The execution of a BT starts from the root node, which ticks its child nodes. A ticked node returns a status of $Success$, $Faliure$ or $Running$ to its parents. *Fallback* node, and *Sequence* node are the commonly used control flow nodes.

Fallback node: It ticks its child nodes from left to right. If a child node returns $Failure$, it ticks the next child node. If a child node returns $Success/Running$, it returns $Success/Running$ and stop ticking. Only all child nodes return $Failure$, does it return $Failure$. A '?' in box is used to represent a *fallback* node.

Sequence node: It ticks its child nodes from left to right. If a child node returns $Success$, it ticks the next child node. If a child node returns $Failure/Running$, it returns $Failure/Running$ and stop ticking. Only all child nodes return $Success$, does it return $Success$. A '$\rightarrow$' in box is used to represent a *sequence* node.

Execution nodes are categorized as two kinds:

Action node: a ticked *action* node perform an action of robot. It returns $Success/Failure/Running$, indicating the action is finished/failed/being executed. A rectangle is used to represent an *action* node.

Condition node: a ticked *condition* node checks whether a condition is satisfied. It returns $Success/Failure$, indicating the condition is true/false. An ellipse is used to represent a *condition* node.

The execution mechanism of action nodes in BTs is similar to PDDL [20]. When an *action* node in a BT needs to be

executed, some *condition* (known as pre-conditions) nodes must be satisfied firstly. Once an action node is completed, some other condition (known as post-conditions) nodes are then satisfied. The paradigm of an action node can be defined as Formula 1.

$$\begin{aligned} a &= \{\texttt{Pre}(a), \texttt{Post}(a)\} \\ \texttt{Pre}(a) &= \{c_1^{pre}, ..., c_n^{pre}\} \\ \texttt{Post}(a) &= \{c_1^{post}, ..., c_m^{post}\} \end{aligned} \tag{1}$$

Where a represents an *action* node, c represents a *condition* node. $\texttt{Post}(a)$ indicates the post-conditions of a, $\texttt{Pre}(a)$ indicates the pre-conditions of a.

IV. The Proposed Method

In this section, we introduce the modules in LLM-BT, which are Recognition, Reasoning, Parser and BTs Update.

A. Recognition

Fig. 2 is the pipeline of Recognition. Firstly, point cloud data from the real-time scene is captured by a 3D camera. We proposed a 3D object recognition algorithm [21] to obtain information of objects, such as their ID, name, color, shape, and 3D spatial coordinates. Subsequently, this information is written in a semantic map as XML format.

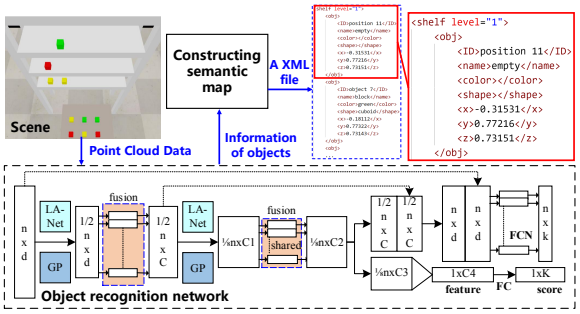

Fig. 2. Pipeline of Recognition.

We have chosen XML as the format for storing the semantic map because of its excellent extensibility, clear structure, and its ability to effectively represent relationships between different attributes.

B. Reasoning

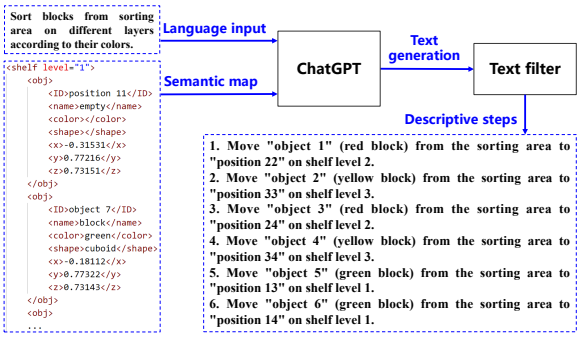

Fig. 3. Pipeline of Reasoning.

In Reasoning module, we utilize the reasoning ability of ChatGPT [14] to understand the information from semantic map and user input. Due to the text generated by ChatGPT not only include descriptive steps, but also many explanatory or declarative statements, a text filter is designed to prevent interference with subsequent keyword extraction. Fig. 3 shows the pipeline of Reasoning.

C. Parser

Parser module consists of two parts, which are *keywords extraction* and *keywords parsing*. The *keywords extraction* uses a BERT-based [15] LLM to extract keywords. Fig 4 shows the pipeline of *keywords extraction*. The descriptive steps are first split by a tokenizer into a set of tokens as inputs of the LLM. Then, the tokens which are labeled are regarded as keywords.

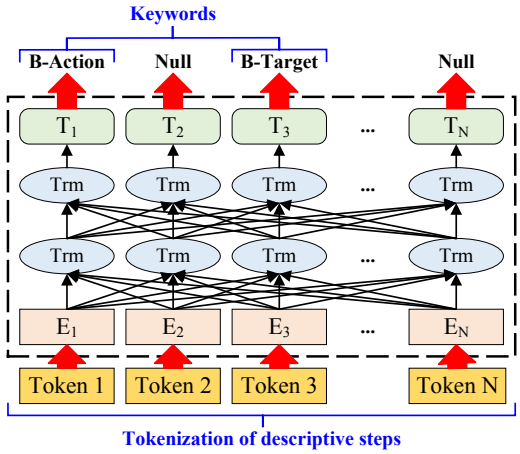

Fig. 4. Pipeline of *keywords extraction*.

During pre-training, we designed six types of labels in datasets, which are 'B-Action', 'B-Target', 'I-Target', 'B-Destination', 'I-Destination', 'B-Location' and 'I-Location'. 'B-Action' represents a token is the name of an action. 'B-Target' or 'I-Target' represents a token is the name of an operating objective. 'B' means the first word of the name and 'I-Target' means the subsequent word of the name. Similarly, 'B-Destination' or 'I-Destination' represents a token is the name of a destination for operating. 'B-Location' or 'I-Location' represents a token is the name of a location.

Subsequently, these keywords are parsed. Fig. 5 shows the process of *keywords parsing* based on our previous automatic programming [22].

The overall parsing is based on a "if-else" structure. Each keyword will be read in turn. When the label of a keyword w is 'B-Action', w represents an action a. Then, ATL will retrieve an appropriate condition node. The action templates in ATL are designed based on Formula 1. According to a, a condition node c that satisfies $c \in Post(a)$ will be found in ATL.

Then, a condition node $c \in Post(a)$ which can be satisfied after completing a will be found from an Action Template Library (ATL). The action templates in ATL are designed based on Formula 1. So the

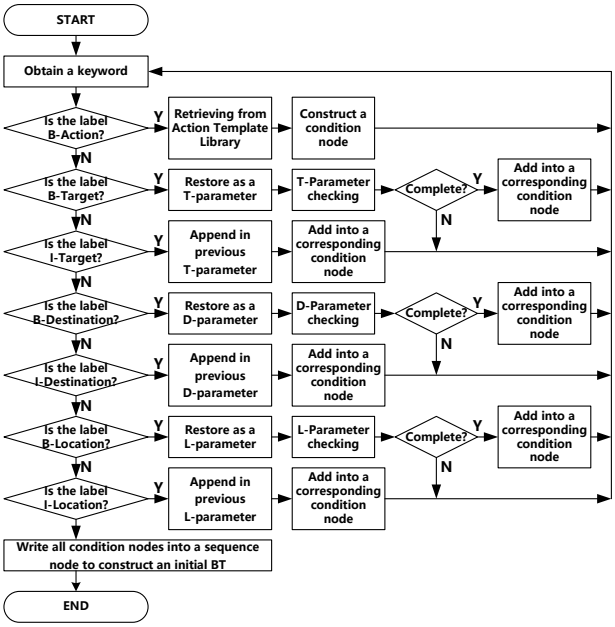

Fig. 5. The process of *keywords parsing*.

When w belongs to 'B-Target',w is a T-parameter (a parameter of the operating objective). Then, w needs to be checked for the integrity of a T-parameter. This is because some parameters include multiple keywords, such as 'position 1' or 'potato chips'. If w is complete, it will be added into a corresponding condition node. otherwise, a new keyword is read for a new round of processing.

When w belongs to 'I-Target', w is a part of previous T-parameter. Then, w is integrated into a T-parameter. And this T-parameter will be added to a corresponding condition node.

Similarly, D-parameter (a parameter of the destination for operating) and L-parameter (a parameter of the location) are obtained and added to corresponding condition nodes.

When all keywords have been traversed, it writes all conditions node into a sequence node to construct an initial BT which represents the goal of a task. Fig. 6 shows the initial BT constructed by Parser based on the descriptive steps in Fig. 3.

Fig. 6. An initial BT constructed by Parser.

Due to the manipulator is fixed in this scene (Fig. 1), only T-parameters and D-parameters are useful in condition nodes. For mobile manipulation robots, L-parameters are required (refer to experiments of household service in section V.B the home service experiment).

D. BTs Update

1) The process of BTs Update: BTs Update mainly includes two functions: `Expand` and `Insert`. Algorithm 1 shows its pseudo-code. Where A represents a set of *action* nodes and C represents a set of *condition* nodes. $\mathcal{T}$ denotes a BT or a subtree and Γ represents a set of subtrees. R is the returned status of a BT.

Algorithm 1: BTs Update

1 **while** $R \neq Success$ **do**
2 $\quad R \leftarrow \texttt{Execute}(\mathcal{T})$;
3 $\quad$ **if** $R = Failure$ **then**
4 $\quad\quad c_f \leftarrow \texttt{GetFailedNode}(\mathcal{T})$;
5 $\quad\quad \mathcal{T}_{exp}, \texttt{Pre}(a) \leftarrow \texttt{Expand}(c_f)$;
6 $\quad\quad \mathcal{T} \leftarrow \texttt{Insert}(\mathcal{T}, c_f, \mathcal{T}_{exp}, \texttt{Pre}(a))$;
7 **Function** `Expand`(c_f)**:**
8 $\quad \mathcal{T}_{exp} \leftarrow \emptyset$;
9 $\quad \Gamma \leftarrow \emptyset$;
10 $\quad$ **for** a **to** A_{tp} **do**
11 $\quad\quad$ **if** $c_f \in \texttt{Post}(a)$ **then**
12 $\quad\quad\quad \mathcal{T}_{st} \leftarrow \mathsf{SequenceNode}(\texttt{Pre}(a), a)$;
13 $\quad\quad\quad \Gamma \leftarrow \mathcal{T}_{st}$;
14 $\quad \mathcal{T}_{exp} \leftarrow \mathsf{FallbackNode}(c_f, \Gamma)$;
15 $\quad$ **return** $\mathcal{T}_{exp}, \texttt{Pre}(a)$;
16 **Function** `Insert`$(\mathcal{T}, c_f, \mathcal{T}_{exp}, Pre(a))$**:**
17 $\quad C_h \leftarrow \texttt{HigherPriorityNodes}(\mathcal{T}, c_f)$;
18 $\quad C_a \leftarrow \texttt{Pre}(a)$;
19 $\quad C_t \leftarrow \texttt{Type}(C_h) \cap \texttt{Type}(C_a)$;
20 $\quad$ **if** $C_t = \emptyset$ **then**
21 $\quad\quad \mathcal{T} \leftarrow \texttt{Replace}(\mathcal{T}, c_f, \mathcal{T}_{exp})$;
22 $\quad$ **else**
23 $\quad\quad \mathcal{T} \leftarrow \texttt{Add}(\mathcal{T}, C_t, \mathcal{T}_{exp})$;
24 $\quad$ **return** $\mathcal{T}$;

At first, an initial BT is executed. When $\mathcal{T}$ returns $Failure$ after being executed, the *condition* node c_f that leads to the failure of $\mathcal{T}$ will be obtained by `GetFailedNode` (lines 3-4 of Algorithm 1). Then, `Expand` constructs an expanded subtree $\mathcal{T}_{exp}$ and insert it into $\mathcal{T}$(lines 5-6). The purpose of `Expand` is to change c_f from $Failure$ to $Success$ by performing new *action* nodes.

If $\mathcal{T}$ still returns $Failure$, $\mathcal{T}$ will be expanded again. Therefore, BTs Update incrementally expands $\mathcal{T}$ to satisfy each failed node in $\mathcal{T}$.

2) Expand: `Expand` searches for appropriate actions in ATL A_{tp}. When there is an *action* node a whose post-conditions contains c_f, it means that the completion of a will change c_f from $Failure$ to $Success$ (lines 10-11 of Algorithm 1).

Then, a subtree $\mathcal{T}_{st}$ will be constructed to ensure the pre-conditions of a are satisfied before executing a. Next, $\mathcal{T}_{st}$ is collected by Γ (lines 12-13 of Algorithm 1). Finally, a *fallback* node will be set as the parent of c_f and Γ to

construct a expanded subtree (lines 14-15 of Algorithm 1).

Here, SequenceNode constructs a subtree with a *sequence* node as its top node. The children of it can be a node (a or c) or a subtree ($\mathcal{T}$). Similarly, FallbackNode constructs a subtree with a *fallback* node as its top node.

When a is completed, c_f may be satisfied. And when c_f returns $Success$, other nodes in $\mathcal{T}_{exp}$ will no longer be traversed. Fig. 7 is the process of `Expand`, it gives a comprehensive solution for c_f.

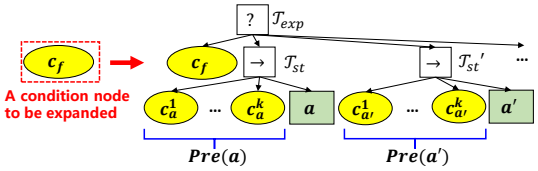

Fig. 7. The process of `Expand`.

3) Insert: `Insert` analyzes the conflict between $\mathcal{T}$ and $\mathcal{T}_{exp}$ and then inserts $\mathcal{T}_{exp}$ into a appropriate position in $\mathcal{T}$. The *condition* nodes in $\mathcal{T}$ with higher priority than c_f (rank before c_f) are collected by `HigherPriorityNodes`. If set C_h and set C_a do not contain nodes of the same type, it denotes that there is no conflict between $\mathcal{T}$ and $\mathcal{T}_{exp}$. Then, c_f is replaced by $\mathcal{T}_{exp}$ in $\mathcal{T}$ (lines 19-21 of Algorithm 1).

If C_h and C_a contain nodes of the same type, replacing c_f by $\mathcal{T}_{exp}$ directly will cause a conflict. This is because the purpose of $\mathcal{T}_{exp}$ is to perform a new action. If there is a node in C_h returns $Success$, a node in C_a will never be satisfied. Fig. 8 is an example of conflict.

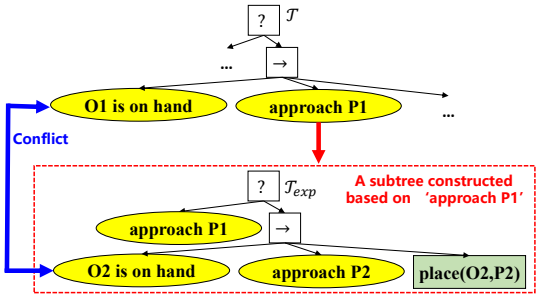

Fig. 8. An example of conflict.

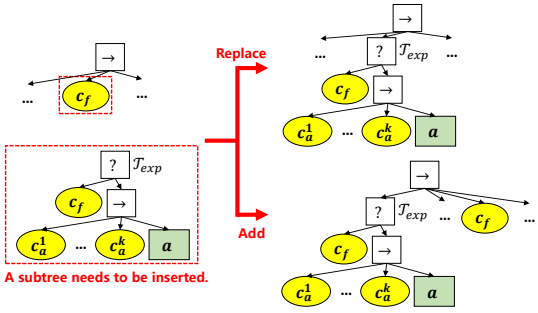

Fig. 9. The process of `Insert`.

At this time, $\mathcal{T}_{exp}$ is added before all nodes in C_t for raising the priority of T_{exp} (lines 22-23 of Algorithm 1). Fig. 9 gives the process of `Insert`.

V. Experiments

We conduct two kinds of experiments which are Cargo sorting and Household service. In Cargo sorting, a fixed manipulator needs to sort cargo according to the user's request. In Household service, a mobile manipulation robot needs to bring objects required for user.

A. Cargo sorting

There are 5 different cases designed in Cargo sorting. Each case invites 20 non-professional users to communicate with the robot for informing their request of sorting tasks (similar to Fig. 1). Meanwhile, there is a random external disturbance (such as an object drops from the manipulator or is blocked by an obstacle) deployed every time the robot works.

To control the variables, the success rates of actions and 3D recognition are not taken into account.

TABLE II
THE RESULTS OF CARGO SORTING.

Cases	Overall (S/ALL)	Reasoning (S/ALL)	Parser (S/ALL)	BTs Update (S/ALL)
Case 1	18/20	18/20	18/18	18/18
Case 2	17/20	18/20	17/18	17/17
Case 3	16/20	17/20	17/17	16/17
Case 4	18/20	18/20	18/18	18/18
Case 5	18/20	19/20	18/19	18/18

Table II shows the results. Where 'Overall' represents the overall success rate of Cargo sorting. 'S/ALL' represents the number of successes out of 20 tests. A success is considered respectively when 'Reasoning' deduces a set of correct descriptive steps, 'Parser' constructs a correct initial BT, and 'BTs Update' returns $Success$.

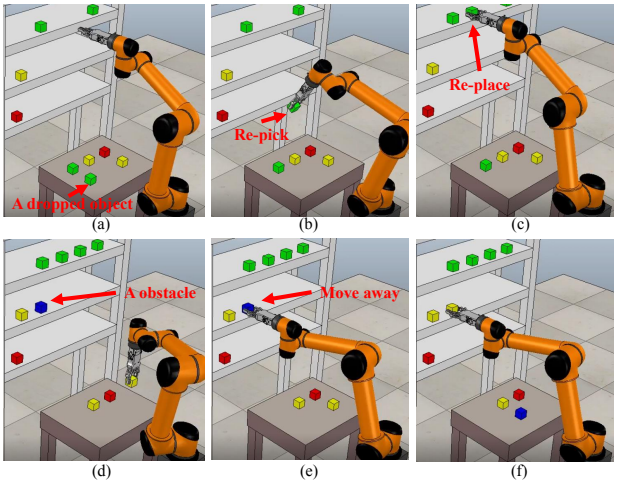

Fig. 10. Dealing with external disturbances during sorting tasks.

The success rate of Cargo sorting was about 85%. In these 5 cases, there were 10 failures caused by unclear requirements expressed by users, which led to the incorrect descriptive steps deduced by Reasoning module. There was

1 failure caused by a unencountered keyword, so that the Parser module constructed an incorrect initial BT. There was 1 failure caused by an unsolvable external disturbance, which BTs Update returned $Failure$.

Apart from these failures, the robot was able to achieves robotic adaptive tasks. As shown in Fig. 10, the robot could pick up an object again when the object dropped from the end effector (Fig. 10(a)(b)(c)). It could move an obstacle away when it hindered a picking or placing action (Fig. 10(d)(e)(f)).

B. Household service

There are also 5 different cases designed in Household service. Each case invites 20 non-professional users to communicate with the robot for informing their request. There is also a random external disturbance deployed every time the robot works.

TABLE III

THE RESULTS OF HOUSEHOLD SERVICE.

Cases	Overall (S/ALL)	Reasoning (S/ALL)	Parser (S/ALL)	BTs Update (S/ALL)
Case 1	15/20	17/20	16/17	15/15
Case 2	14/20	16/20	14/16	14/14
Case 3	15/20	16/20	15/16	15/15
Case 4	16/20	18/20	16/18	16/16
Case 5	17/20	17/20	17/17	17/17

Table III shows the results. The success rate of Household service decreased to about 75%. With the increasing complexity of environment, user's unclear expressions could lead to a higher probability of reasoning failure (16 out of 100 tests). The complicated environment also led to the generation of novel descriptive steps, which caused failure of Parser module (6 out of 84 tests). Therefore, expanding the dataset of BERT-based LLM is an important part for improving LLM-BT. Apart from these failures, the robot was able to achieves robotic adaptive tasks, as shown in Fig. 11

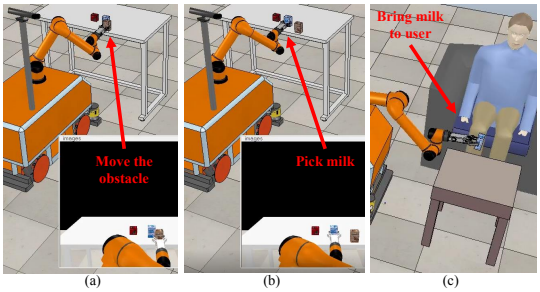

Fig. 11. Dealing with external disturbances during household service. (a) the robot firstly moved the obstacle away from the milk, (b) picked the milk, (c) brought it to user.

Consequently, the results of these two experiments demonstrate the feasibility of our method. We demonstrate the video[2] of Cargo sorting and Household service.

[2]https://youtu.be/TkIWb53IEtQ

VI. DISCUSSION

A. Advantages

Compared to other LLM-based methods, the notable advantage of our method is adaptability. LLM-BT is capable of adding new actions through ATL based on environmental changes and assigning appropriate executing priorities to these actions.

The environmental changes may result in different operations for the same task. We consider that an effective method is to utilize LLM to deduce the step of a task. Then, paring it into a set of conditions that need to be satisfied. And finally using BTs Update algorithm to execute appropriate actions based on the current environment.

B. Limitations

Although LLM-BT demonstrates adaptability, there are two limitations in our method.

(1) Relatively weak understanding of scene: The system is unable to learn new knowledge from the semantic map. For example, in cargo sorting experiments, the system only places each kind of object on the empty position of corresponding layer, without arranging them in a specific order. This is because it cannot infer the relative positions of objects based on their 3D spatial coordinates.

(2) Manually construct ATL: the ATL is a crucial component for Parser module and `Expansion` in BTs Update. The action templates in the ATL are added manually by experts or engineers. When the system needs to add a new kind of action node, a corresponding action template also needs to be added.

VII. CONCLUSION AND FUTURE WORK

In this paper, LLM-BT is proposed. It utilizes LLMs to construct initial BTs automatically, and then uses BTs Update algorithm to expand them dynamically. The experiments demonstrate that LLM-BT enables robots to perform robotic adaptive tasks.

In the future work, we will focus on improving two parts of LLM-BT. The first part is to improve the parsing accuracy of keywords in the operation parsing module by utilizing an advanced deep learning model. The second part is to apply robotic manipulation algorithms for deformable objects to conduct complicated experiments.

REFERENCES

[1] T. Brown, B. Mann, N. Ryder, et al, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] A. Bucker, L. Figueredo, S. Haddadinl, et al, "Reshaping robot trajectories using natural language commands: A study of multi-modal data alignment using transformers," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Kyoto, Japan, 2022, pp. 978–984.

[3] W. Huang, F. Xia, T. Xiao, et al, "Inner monologue: Embodied reasoning through planning with language models," *arXiv preprint arXiv:2207.05608*, 2022.

[4] D. Driess, F. Xia, M. S. M. Sajjadi, et al, "Palm-e: An embodied multimodal language model," *arXiv preprint arXiv:2303.03378*, 2023.

[5] A. Brohan, N. Brown, J. Carbajal, et al, "RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control," *arXiv preprint arXiv:2307.15818*, 2023.

[6] S. Vemprala, R. Bonatti, A. Bucker, et al, "Chatgpt for robotics: Design principles and model abilities," *Microsoft Auton. Syst. Robot. Res*, vol. 2, pp. 20, 2023.
[7] M. Ahn, A. Brohan, N. Brown, et al, "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.
[8] W. Huang, C. Wang, R. Zhang, et al. "VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models," *arXiv preprint arXiv:2307.05973*, 2023.
[9] M. Colledanchise, P. Ögren, *Behavior trees in robotics and AI: An introduction*, Boca Raton, Florida, United states, CRC Press, 2018.
[10] S. Rabin, *Game AI pro 3: collected wisdom of game AI professionals*, Boca Raton, Florida, United states, CRC Press, 2017, pp. 1–514.
[11] M. Colledanchise, L. Natale, "On the implementation of behavior trees in robotics," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5929–5936, 2021.
[12] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 372–389, 2016.
[13] M. Colledanchise, D. Almeida and P. Ögren, "Towards blended reactive planning and acting using behavior trees," *International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada, 2019, pp. 8839–8845.
[14] Y. Liu, T. Han, S. Ma, et al, "Summary of chatgpt/gpt-4 research and perspective towards the future of large language models," *arXiv preprint arXiv:2304.01852*, 2023.
[15] J. Devlin, M. W. Chang, K. Lee K, et al, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
[16] A. Brohan, N. Brown, J. Carbajal, et al, "Rt-1: Robotics transformer for real-world control at scale." *arXiv preprint arXiv:2212.06817*, 2022.
[17] K. Y. W. Scheper, S. Tijmons, C. C. de Visser and et al, "Behavior trees for evolutionary robotics," *Artificial life*, vol. 22, no. 1, pp. 23–48, 2016.
[18] K. French, S. Wu, T. Pan and et al, "Learning Behavior Trees From Demonstration," in *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada, 2019, pp. 7791–7797.
[19] B. Banerjee, "Autonomous acquisition of behavior trees for robot control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, 2018, pp. 3460–3467.
[20] M. Fox, D. Long, "PDDL2. 1: An extension to PDDL for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.
[21] Y. Lin, T. Liu, Y. Zhang, et al, "LA-Net: LSTM and attention based point cloud down-sampling and its application," *Measurement and Control*, pp. 1–17, 2023.
[22] Y. Lin Y, H. Zhou, M. Chen, et al, "Automatic sorting system for industrial robot with 3D visual perception and natural language interaction," *Measurement and Control*, vol. 52, no. 1-2, pp. 100-115, 2019.