2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

DiaGBT: An Explainable and Evolvable Robot Control Framework using Dialogue Generative Behavior Trees

Jinde Liang[1,2], Yuan Chang[2], Qian Wang[2], Yanzhen Wang[2] and Xiaodong Yi[2]

Abstract— Manipulating robots using natural language is the preferred way for non-technical specialists. The challenge lies in reliability and adaptability especially when robots operate in unstructured surroundings. In this paper, we propose a novel framework called Dialogue Generative Behavior Trees (DiaGBT). Natural language instructions from human operators are transformed into behavior trees (BTs) and further executed by robots. Compared to the emerging Large Language Models (LLMs), DiaGBT is comparable in terms of semantic understanding but more lightweight, since the parsing rules are produced by LLM but tailored for task-correlated instructions. Besides, DiaGBT allows multi-round human-robot interaction, where robots learn reusable skills in real time. For evaluation, we generate a dataset with 4k instruction-BT pairs covering 4 different scenarios. On average, DiaGBT reaches over 90% parsability and 80% plausibility. Similar results on the VEIL-500 dataset outperform the current state of the art.

I. INTRODUCTION

It is now evident that robots will be ubiquitous in the future [1]. They will serve as indispensable counterparts to supplant firefighters in searching and rescuing or to aid elderly people in their daily lives, as illustrated in Fig. 1. In these scenarios, robots must interact with non-technical specialists in real time to accomplish tasks within unstructured environments. Despite many prototypes of chat-instructed robots, we still face significant challenges in reliability and adaptability that have yet to be resolved [2].

First, the behavior of robots must be explainable and consistent with human intentions [3]. This excludes some end-to-end solutions as they are typically black boxes. In this regard, the behavior tree (BT) is favored as an interpretable representation for robot tasks [4]. A BT is a tree with behaviors described by leaf nodes and logic formulated by control nodes. BTs were invented for games but have received an increasing amount of attention in the robotics community due to their modularity and interpretability [4].

Unfortunately, most earlier works on BTs mainly focus on extending their flexibility [5][6] while generating BTs from natural language has only recently become a focus [7][8][9]. In [7], Gavin et al. developed Lingua, which demonstrated the potential of learning and executing reusable BTs from human instructions. However, Its accuracy is undermined mainly by hand-crafted grammar.

[1]University of Electronic Science and Technology of China (UESTC), Chengdu, 611731, China
[2]Intelligent Game and Decision Lab (IGDL), Beijing 100071, China

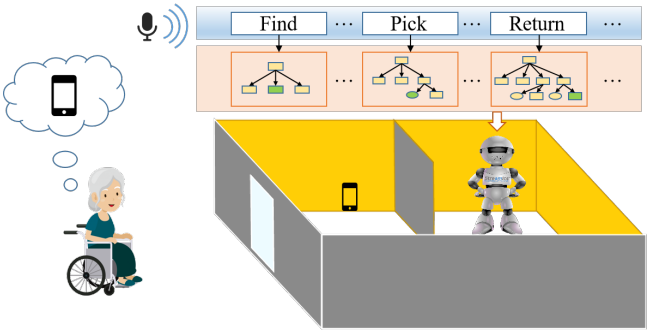

Fig. 1: An example of an old lady telling a robot to fetch a phone from an inaccessible room. The language instructions of the lady are transformed into BTs to drive the robot to execute sequential actions.

Not long after, the bottleneck of semantic understanding was broken through by Large Language Models (LLMs) [10], indicating new opportunities for robot control. In [8], Cao and Lee proposed the use of the Generative Pre-trained Transformer (GPT) for generating BTs. A similar work was presented by Lykov and Tsetserukou [9], except that the LLM was replaced with a self-trained LLM-BRAIn.

Second, the robot should have the ability to learn new skills to adapt to diverse tasks [11]. This also can be supported by LLM through the prompt-based learning paradigm [12]. In [13], an abstract task descriptor and its associated basic task sequence were used to set the prompt. The SayCan framework [14] combined human high-level instructions and corresponding robot basic tasks as prompts. In [15], Feifei et al. combined LLM and visual models to form a value map by extracting affordances and constraints to synthesize the trajectory of robot operations. These works show the capability of LLM for task planning but have limitations when robot tasks shift to unfamiliar surroundings.

In sum, the LLM-empowered methods mentioned above directly utilize the LLM as a brain to understand human instructions and to generate task plans. However, the operation of LLMs typically requires sufficient computing resources, prohibiting their usage in applications with limited communication bandwidth or GPUs. Moreover, since the LLM is data-driven, its performance highly depends on the quality of training data. Therefore, this paradigm does not meet the requirements for reliability and adaptability.

In this paper, we propose a novel framework called Dialogue Generative Behavior Trees (DiaGBT). Different from the above methods, DiaGBT uses LLM as the machine tool to produce rule components for BT generation [16]. As such, DiaGBT combines the advantages of Lingua's interpretability and LLM's powerful semantic understanding. It is comparable to LLM in terms of instruction parsing but is more lightweight. It can be deployed offline or online, thus suitable for a variety of applications. Moreover, we design a human teaching module to meet the individual needs of end-users. The teaching module generates new BTs in response to new tasks and the learned skills are stored in a BT memory for future re-usage.

The contributions of this paper are as follows.

- We developed an easy-to-use robot control framework called Dialogue Generative Behavior Trees (DiaGBT). The framework transforms human instructions to BTs in an explainable way, allowing real-time and trustworthy human-robot collaboration.
- We proposed a new paradigm for LLM-empowered BT generation. This paradigm uses LLM as the machine tool to produce components for a smaller model. In this way, DiaGBT inherits LLM's embedded knowledge for semantic understanding but is more lightweight.
- We designed a teaching module for the robot to resolve ambiguous instructions and learn new skills. This endows robots with the ability to generate the correct BT, meet the particular needs of users, and evolve for a broader range of tasks.

The code and dataset for this paper are open source.[1]

II. BACKGROUND

A. Behavior Tree

The BT is a hierarchical representation of robot tasks. A BT example is given in Fig. 2. Note that the BT has one root node, some control nodes as non-leaf nodes, and some execution nodes as leaf nodes. The execution of a BT starts from the root node, which generates signals called Ticks with a given frequency. A node that is ticked will return Running (R) if its execution is underway, Success (S) if it has achieved its goal, or Failure (F) otherwise. The functions of typical BT nodes are briefly introduced as follows.

Sequence: When a Sequence is ticked, it sends ticks to its children in sequential order. If all the children nodes return S, then S is returned upwards. If one of its children nodes returns F, the Sequence directly returns F.

Fallback: The Fallback differs from Sequence in that as soon as one child returns S, it returns S directly. The node returns F only when all children return F.

Parallel: The Parallel ticks multiple nodes simultaneously. If M out of the N children returns S, then so does the Parallel. If more than $N - M$ children return F, it returns F.

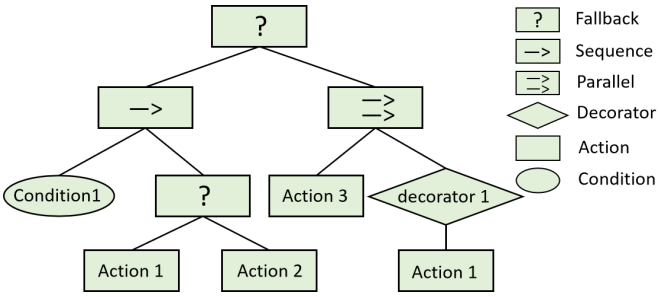

Fig. 2: An example of a behavior tree.

Decorator: The decorator is used for looping, negating, and other operations on leaf nodes.

Condition: The Condition check a proposition upon receiving Ticks and return S or F depending on if the proposition holds or not.

Action: Action is used to execute a command and returns S if the action is completed.

B. Large Language Model in Robotics

The tremendous success of LLMs in the NLP field has aroused great interest worldwide [10]. Since they are trained in a task-agnostic manner and can be adapted to a wide range of downstream tasks, there is no doubt that the robotic community will be largely impulsed. The LLMs have shifted the traditional "fine-tune" paradigm to the new "prompt-based learning" paradigm [17]. This paradigm modifies the original input x into a *Prompt* template with some unfilled slots and then calls the LLM to generate an output of sequence, i.e. $x_p = f_{prompt}(x)$, where template x_p consists of an input slot, an output slot, and additional prompt information. In the prompt-based learning paradigm, prompt design plays a key role [18].

Currently, early LLM-based robots typically use the prompt-based learning paradigm, as introduced above [8][9]. However, since it is too expensive to train a LLM for each robot or each task, a more economical way would be to use LLMs to produce essential components for robot control. In other words, the LLM is used as a machine tool, which is the focus of this paper.

III. METHODS

A. Framework Overview

We use the BT as the bridge of human-robot collaboration. Two problems need to be solved to translate NL instructions to BTs. The first problem is how to generate a BT that confirms human intentions (Problem 1). This requires extracting BT elements from the NL instruction and correctly combining them. In addition, to eliminate ambiguities in NL instructions, multiple conversations between the operator and the robot are essential. The second problem is how to modify the executing BT to

[1][Online].Available: https://github.com/jayden-leo/dia-gbt

control the robot in motion (Problem 2). This requires additional contextual considerations.

To address these two issues, we design a framework for dialogue generative behavior trees (DiaGBT), as Fig. 3 shows. The BT Generation Loop and the BT Execution Loop correspond to Problem 1 and Problem 2, respectively. In terms of execution timing, the BT execution loop is the interval between two adjacent BT generation loops. The BT execution loop represents the running life cycle of a task. During the execution process, the BT is modified in real-time through multiple rounds of BT generation loops.

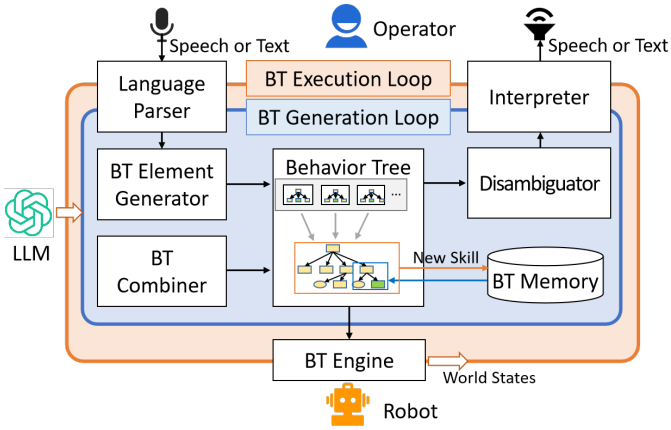

Fig. 3: System architecture of the DiaGBT framework.

The main blocks of DiaGBT are briefly introduced as follows. Starting from natural language input from the operator, the Language Parser and the Interpreter involve basic operations related to speech, such as tokenization, word vector embedding, text-to-speech encoding, etc. The BT Element Generator is used to extract BT elements from instructions, their dependencies are described by the BT Combiner. The Disambiguator is used to feed back ambiguous phrases to the operator for further clarification. The BT Memory is used to store learned skills for future re-usage. The generated BT is output to the BT Engine to execute tasks.

DiaGBT is rule-based and the rules are drawn from the LLM. To achieve this, we adopt a method of online training and offline deployment. During online training, the system obtains intent and mapping knowledge from the LLM. During offline deployment, the system runs based on learned rules and human-robot interaction.

B. Task-correlated Instructions

Before delving into the details of BT generation, it is necessary to clarify the boundaries of the NL instruction set [19]. Note that imperative instructions are what we truly care about in robot control, rather than aimless chatting. This principle is beneficial for parsing and interpretation since imperative instructions typically have clear and concise syntax. Interestingly, This constraint even increases the user experience with a more precise response [20].

Consider the task space S, each BT $T \in S$ stands for an executable task, which is constrained by robot capabilities and environments. Due to the subjectivity of language, T may have multiple descriptions. The set of these descriptions is denoted as O_T. Therefore, the NL instruction set can be defined as $O = \bigcup O_T$ for all $T \in S$.

Note that NL instructions and BTs are not mapped one by one. An instruction may correspond to multiple BTs. As such, we divide O into two categories. If $o_i \in O$ can be formulated by a unique BT, then we call it a type A instruction, i.e. $o_i \in O_A$, otherwise it is called a type B instruction, i.e. $o_i \in O_B$.

Type A instructions describe detailed and unambiguous task steps. For example, "Go to the kitchen and get a bowl". A direct description of the BT also falls into this category. For example, "Plan a parallel task with two children, searching and mapping". The latter requires some knowledge of the BT but is useful when accurate input is required. According to combinatory categorial grammar (CCG) [21], we have $O_A = \{C, R\}$, where C represents the basic syntactic categories such as nouns, verbs, adjectives, etc. R represents the function composition of the categories.

Type B instructions are more concise, only providing simple descriptions of the task and overlooking some implementation details. For example, "Clean the house". A type B instruction must be decomposed into a series of executable subtasks, i.e. type A instructions, before being executed. Therefore, mapping rules from O_B to O_A are required, and we store these rules by BTs in the BT Memory. The rules can be taught by LLM or via human interaction in the Disambiguator. Fig. 4 reveals the two-stage workflow of DiaGBT in processing two types of NL instructions. Details of BT generation are presented in the following subsections.

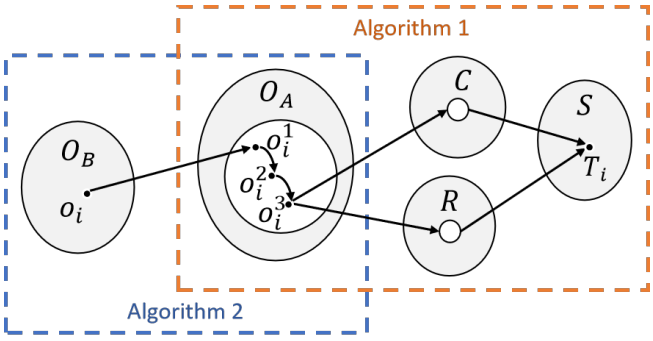

Fig. 4: Mapping from NL instructions to BTs. Algorithm 1 and Algorithm 2 handle type A instructions and type B instructions, respectively.

C. BT Element Generation and Composition

We first consider dealing with type A instructions. The relevant functional modules in the DiaGBT framework include the Language Parser, the BT Element Generator,

and the BT Combiner. To acquire the correct BT, we need to extract BT elements from the instruction and combine them in the right manner. The method is summarized in Alg. 1.

Algorithm 1: Mapping type A instructions to BTs.

Input : Type A instruction $o_i \in O_A$
Output: BT $T_i \in S$ that conforms to o_i

$C = \{c_1, c_2, ..., c_n\} \leftarrow \text{Parse}(o_i)$;
$T' = \{T'_1, T'_2, ..., T'_m\} \leftarrow$ List_all_combinations(C);
$O'_A = \{o'_1, o'_2, ..., o'_m\} \leftarrow \text{Translate}(T')$;
for $j \leftarrow 1$ to m do
  $v'_j \leftarrow \text{Encode}(o'_j)$
end
$v_i \leftarrow \text{Encode}(o_i)$;
$k \leftarrow \arg\max_{j \in [1,m]} (v'_j \cdot v_i)$;
$T_i \leftarrow T'_k$;

The first line of Alg. 1 is used to extract BT Elements C by parsing o_i. A BT element is defined as a condition node or an action node. The NL expression that corresponds to a BT element is grammatically identifiable. For example, if the NL instruction takes the grammar pattern of "Verb + Noun + if/when/unless + Clause", the "Verb + Noun" is formalized as an action node whereas the "Clause" is formalized as a condition node. We collect these grammar patterns through LLM [22] due to its compression of massive language information and superior ability of semantic understanding.

Directly extracting combination rules from the NL instruction faces great challenges. Instead, we obtain the correct combination of BT elements through enumeration and comparison, corresponding to the second and third rows in Alg. 1. First, we list all combinations of BT elements to get a candidate BT set T'. The reason why violent enumeration is feasible is due to the position limitation of each BT node during the combination process, which process can be pruned at multiple levels. For example, action nodes and condition nodes must exist in the form of leaf nodes, etc. In addition, according to user habits, a single instruction often contains a limited number of BT($n \leq 8$), so it is computationally feasible. Then, we translate T' back to NL instruction set O'_A. The instruction o'_j corresponding to the correct BT must be semantically closest to the original instruction o_i. Therefore, we use a semantic vector encoder to compare them in the semantic space. The BT T'_k corresponding to the candidate semantic vector with the largest inner product is taken as the output.

In addition to specifying a task process in advance, operators often make adjustments to the task during the process. As such, the current BT must be re-edited based on logical dependencies between adjacent NL instructions o_i and o_{i+1}. In this case, we first generate a BT T_{i+1} for o_{i+1} using Alg. 1 and then merge it with T_i. Some common dependencies include juxtaposition, conditionality, causality, contrast, progression, transitivity, interpretation, etc. Each corresponds to a different merging rule, either adding or deleting branches. Similarly, we collect these rules through LLM and store them for retrieval.

D. Learning New Skills from Human

Now we consider dealing with type B instructions. This type of instruction only describes the task objectives and requires additional knowledge to supplement specific steps. In this paper, we design a mechanism for robots to learn new skills from humans. The relevant modules in the DiaGBT framework include the Interpreter, the Disambiguator, and the BT Memory. The method is summarized in Alg. 2.

Algorithm 2: Mapping type B instructions to BTs.

Input : Type B instruction $o_i \in O_B$
Output: BT $T_i \in S$ that conforms to o_i

Initialize equivalent set $\hat{o} = \{o_i\}$;
while $hasTypeB(\hat{o})$ do
  for each element $\hat{o}_j$ in $\hat{o}$ do
    if $isTypeB(\hat{o}_j)$ then
      $\{\hat{o}^1_j, \hat{o}^2_j, ..., \hat{o}^m_j\} \leftarrow$ Ask_human($\hat{o}_j$);
      Replace $\hat{o}_j$ with $\{\hat{o}^1_j, \hat{o}^2_j, ..., \hat{o}^m_j\}$;
    end
  end
end
Generate T_i with $\hat{o}$ using Alg.1;

As shown in Alg. 2, we need to identify the type of instruction. The BT memory stores all the skills of the robot. It is organized as a knowledge graph. If a BT element cannot be retrieved from the BT memory, the corresponding instruction is a type B instruction. The learning process is designed to be fully autonomous to release humans from the teaching burden. As human-robot interaction progresses, the robot generates BTs in real time.

The BT that represents a new skill will be stored in the BT memory. In this paper, we converge a new BT to a semantic vector to compare it with existing skills semantically. If the difference is over a threshold, it indicates that a new skill needs to be learned. Note that a BT can be generalized to accomplish a type of task. For example, the instruction "pick something'' can be generalized to pick an apple or a book. In this regard, we remove the specifics of the parameters of the BT for behavior pattern abstraction.

E. Disambiguation

A BT can only be successfully executed if it conforms to the robot's abilities and task environment. This

prerequisite is primarily influenced by factors such as language ambiguity and environmental invisibility. In other words, the BT must be verified in the generation phase. Otherwise, the disambiguation module will be invoked to report the conflicts to the human for further clarification.

For verification, we construct a set of action nodes based on the robot's capabilities. Each action node of the BT must be included in the set to ensure that it is executable. Moreover, we update robot states and perceived environmental information in real time and check whether there are conflicts for condition nodes.

The disambiguation module is based on multiple rounds of conversations. Unlike LLM's aimless chatting, we define a series of Q&A templates in advance to eliminate BT conflicts. This solution is more concise, efficient, and lightweight, although not as natural as LLM conversations. As a result, the BT is continuously modified until all conflicts are resolved.

IV. EXPERIMENTS

To demonstrate DiaGBT's capability of understanding human instructions and learning new behaviors, we first build a dataset of NL instructions and then run DiaGBT separately under two conditions: without human feedback and with human feedback.

A. Dataset of NL Instructions

We first construct a dataset with 4,521 (Instruction, BT) pairs for evaluation. The dataset, called Multi-Domain Instruction to BT (MDI-BT), covers 4 different scenarios: manufacturing, packaging, cooking, and cleaning. Each scenario contains more than 1,000 data pairs (including type A and type B instructions). A sample of the dataset is given in Fig. 5. The dataset can be downloaded from the open-source project linked above.

The MDI-BT is generated by LLM with well-designed prompts [23]. To ensure the correctness of the data pair, we use a template to convert the BT into a semantically consistent NL. Only data pairs whose semantic vector similarity reaches a threshold are selected. Furthermore, through iterative instruction generation, manual verification, and prompt modification, we developed a prompt template that allowed us to generate qualified instructions [24]. In this way, we can expand the scale of MDI-BT as needed.

B. BT Generation without Human Feedback

a) Testing on MDI-BT. To quantitatively measure the ability of DiaGBT to generate BTs from NL, we conduct dataset testing on MDI-BT. We consider two step-by-step indicators: 1) parsed, which means the instruction has been successfully parsed to generate a BT, and 2) plausible, which means the generated BT is correct. The plausible indicator is checked through semantic vector similarity. More specifically, we translate a generated BT back to natural language by template mapping and calculate cosine similarity between the original instruction and the reverse translated instruction. A BT is correct if the semantic similarity surpasses a threshold.

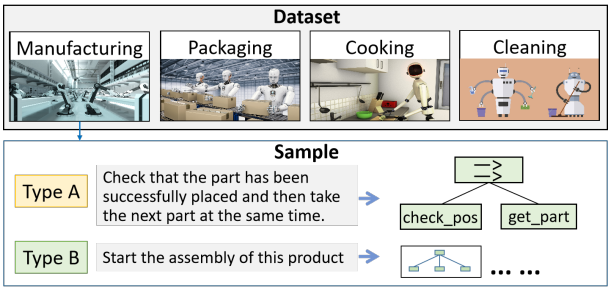

Fig. 5: Samples from the Multi-Domain Instruction to BT (MDI-BT) dataset. Each scenario consists of two types of instructions, namely type A and type B instructions, and their corresponding BTs.

The trial is performed on each scenario either separately (sep) or incrementally (incre). Incremental testing means the acquired rules learned from earlier data can be reserved in the current scenario. The results are shown in Table I. Note that the parsed rate surpasses over 90% regardless of the scenario and initial conditions. The reasons for the failure cases are twofold: either the instruction is too long, or the robot is unable to execute. As for the plausible accuracy, we also notice a high rate of over 80% on average, showing the outstanding performance of DiaGBT. However, a noteworthy observation is that within incremental testing, there is a slight reduction for both the parsed rate and plausible rate. This suggests that the acquired rules from a certain task may also have adverse impacts on other tasks.

TABLE I

PERFORMANCE OF DIAGBT ON MDI-BT

Indicator	Cooking	Packaging		Manufacturing		Cleaning	
	sep	sep	incre	sep	incre	sep	incre
Parsed	98%	98.8%	96.0%	98.1%	91.5%	99.7%	96.7%
Plausible	78.4%	82.7%	79.2%	91.4%	88.0%	87.4%	85.7%

TABLE II

COMPARISON OF DIAGBT WITH LINGUA ON VEIL-500

Lingua	54.21% (baseline)			
DiaGBT	zero-shot	500 samples	1000 samples	useLLM
	4.00%	46.04%	73.90%	96.59%

b) Comparison with VEIL-500 [25]. We also provide comparisons with Lingua, the most related work, and the state of the art. The VEIL-500 dataset used by it contains 2198 natural language instructions, and Lingua achieves a 54.21% parsing rate on this dataset.

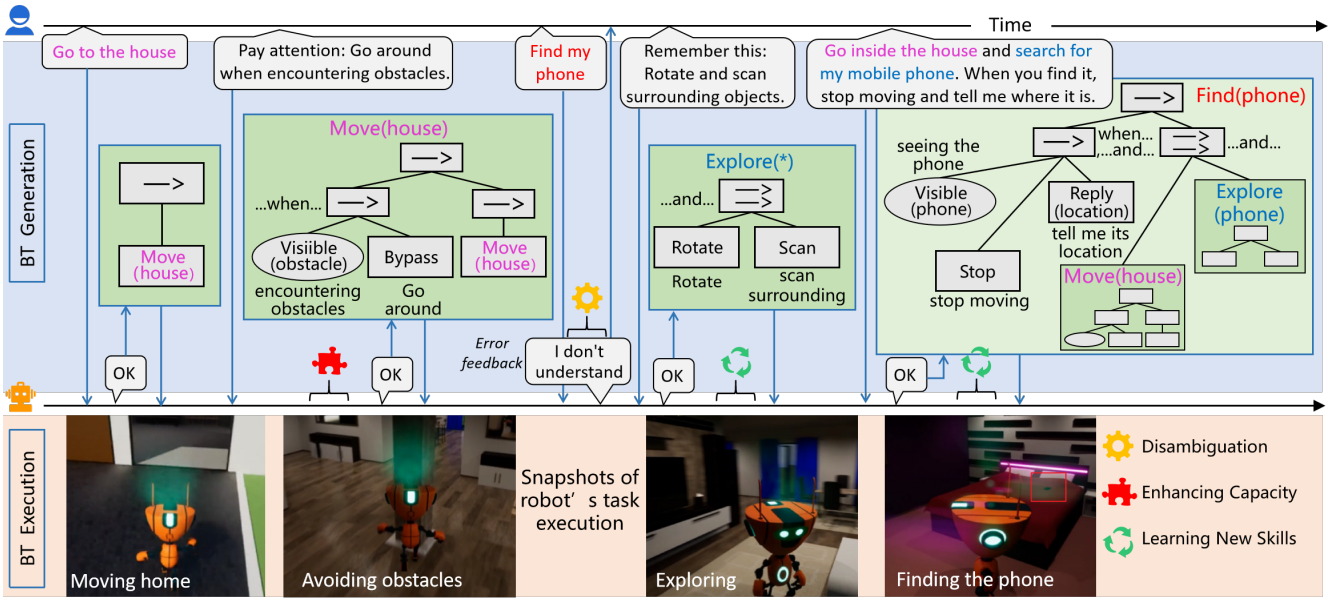

Fig. 6: The workflow of a robot executing "find my phone" task through human-robot collaboration. (Above) Multiple rounds of dialogue during task execution and the underlying process of BT generation. The instructions and BTs are correlated with the same color. (Below) Snapshots of robot's task execution. The states of the robot are visualized by symbols.

The comparison of DiaGBT with the Lingua is shown in Table II. When directly using DiaGBT (learned from MDI-BT) on VEIL-500 in a zero-shot manner, only 4% of instructions can be parsed. The low parsing rate can be attributed to the fact that the VEIL-500 mainly consists of type B instructions, which require additional information either from the LLM or the human operator. Therefore, we further test the online version with LLM connected. First, we let DiaGBT learn from a subset of the VEIL-500 and then perform offline parsing for the remaining instructions. As more samples are used by DiaGBT to familiarize itself with the given scenario, DiaGBT's parsing rate gradually increases until surpasses that of Lingua. Moreover, if we keep LLM connected throughout the parsing process, the parsing rate of DiaGBT is over 96%. The results demonstrate a better performance of our online version compared to the baseline.

c) Task migration capability. We count the number of parsing rules to examine whether DiaGBT has learned generalization knowledge for BT generation. The trial setup is the same as Test a), and the growth trend of learned rules is drawn in Fig 7.

Initially, DiaGBT only had 29 rules. Upon learning, the number of instruction mapping rules stored within the BT Memory gradually increases. After learning from 4521 instructions of MDI-BT, only 512 rules have been stored. This reveals that the learned rule can be generalized to express multiple instructions. Throughout four scenarios, there is an observable trend of convergence in the growth of the rules.

Moreover, we found that compared to separate testing

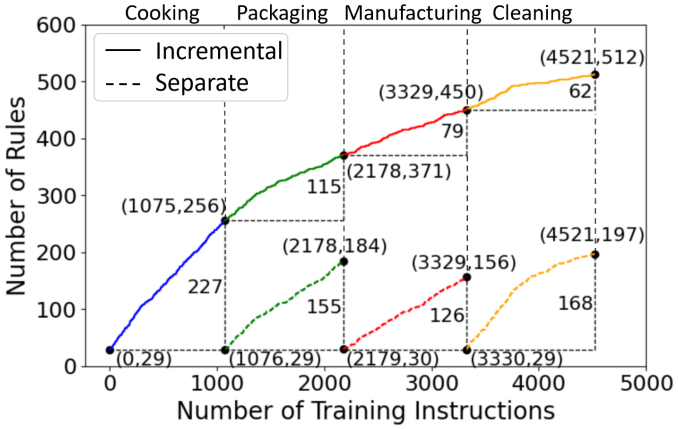

Fig. 7: Growth of the rules of DiaGBT when parsing instructions from the MDI-BT dataset. Each scenario is distinguished by a different color.

on different scenarios, incremental testing result in fewer rules. Take the cleaning task, for example, the number of rules learned during incremental was merely 62, as opposed to 168 in separate testing. This further underscores the reusability of the instruction mapping rules learned from the LLM. It also explains the lightweight nature of DiaGBT, which is essential for robots with limited computing resources.

C. BT Generation with Human Feedback

Human feedback is not only useful for generating BTs correctly but can also make robots more personalized. In this subsection, we demonstrate the ability of DiaGBT to generate a correct BT by learning new skills from humans

in real time. This ability plays a key role in handling type B instructions. To close the gap between BT generation and execution, we developed a simulator using Unreal Engine 5 involving a robot for house service. Unreal Engine 5 integrates a BT engine that ensures automatic task execution with BTs generated by DiaGBT.

An example is given in Fig. 6. During the task, the human operator tells the robot to go to the house and avoid obstacles if encounters them. However, the phrase "Find my phone" cannot be parsed since the robot lacks the necessary knowledge. The disambiguator is called to ask the human operator for Supplementary information. By the guidance of multiple rounds of dialog, a new skill is defined as "Find()", and the robot uses it to fill the BT slot. During the conversation, the system will perform a series of BT generation, runtime modifications, necessary information feedback, reinforcement of existing node capabilities, etc. The learned skill is also stored in the BT memory for future re-usage.

V. DISCUSSIONS and CONCLUSIONS

In this paper, we presented DiaGBT. It combines the strengths of a rule-based model and data-based LLM to achieve excellent BT generation performance with remarkable lightweight features. Similar designs have also been proven effective by [28]. The performance of DiaGBT is demonstrated by dataset testing and case studies. Moreover, DiaGBT can learn reusable skills either from the LLM or from the human operator. This endows robots the ability to evolve to be adaptive and personalized.

Currently, DiaGBT still has some limitations as a prototype. One of the problems is that human-machine dialogue is not as natural as LLMs. This requires more attention to user experience. Another problem is that the BT generation accuracy is sensitive to BT element positions in language instructions. The latest advances in contrastive learning [26][27] provide a powerful method for semantic feature abstraction. It leverages the power of data to achieve a semantic space. Future work will focus on improving the performance of DiaGBT in both accuracy and user experience.

References

[1] F. Huang, P. Liu, Q. Li, X. Yu, "Review: Intelligent control and human-robot interaction for collaborative robots," Chinese Journal of Engineering, vol. 44, no. 4, pp. 780–791, 2022

[2] K. Aggarwal et al, "Deep learning in robotics for strengthening industry 4.0.: opportunities, challenges and future directions," Robotics and AI for Cybersecurity and Critical Infrastructure in Smart Cities, vol. 1030, pp. 1–19, 2022

[3] T. Sakai and T. Nagai, "Explainable autonomous robots: A survey and perspective," Advanced Robotics, vol. 36, no. 5–6, pp. 219–238, 2022

[4] M. Iovino et al, "A survey of behavior trees in robotics and ai," Robotics and Autonomous Systems, vol. 154, pp. 104096, 2022

[5] J. He et al, "Deep reinforcement learning with a natural language action space," 2015, arXiv:1511.04636

[6] Y. Wu et al, "Micros. bt: An event-driven behavior tree framework for swarm robots," in Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, 2021, pp. 9146–9153

[7] G. Suddrey, B. Talbot and F. Maire, "Learning and executing re-usable behaviour trees from natural language instruction," IEEE Robotics and Automation Letters, vol. 7, no. 4, pp. 10643–10650, 2022

[8] Y. Cao and C. Lee, "Robot behavior-tree-based task generation with large language models," 2023, arXiv:2302.12927

[9] A. Lykov and D. Tsetserukou, "LLM-BRAIn: AI-driven fast generation of robot behaviour tree based on large language model," 2023, arXiv:2305.19352

[10] T. Brown et al, "Language models are few-shot learners," Advances in neural information processing systems, vol. 33, no. 68, pp. 1877–1901, 2020

[11] L. Yuan et al, "In situ bidirectional human-robot value alignment," Science robotics, vol. 7, no. 68, pp. eabm4183, 2022

[12] P. Liu et al, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," ACM Computing Surveys, vol. 55, no. 9, pp. 1–35, 2023

[13] W. Huang, P. Abbeel, D. Pathak and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in Proceedings of International Conference on Machine Learning, 2022, pp. 9118–9147

[14] M. Ahn et al, "Do as i can, not as i say: Grounding language in robotic affordances," 2022, arXiv:2204.01691

[15] W. Huang et al, "VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models," 2023, arXiv:2307.05973

[16] V. Viswanathan et al, "Prompt2Model: Generating Deployable Models from Natural Language Instructions," 2023, arXiv:2308.12261

[17] B. Min et al, "Recent advances in natural language processing via large pre-trained language models: A survey," 2021, arXiv:2111.01243

[18] L. Reynolds and K. McDonell, "Prompt programming for large language models: Beyond the few-shot paradigm," in Proceedings of Extended Abstracts of the Conference on Human Factors in Computing Systems, 2021, pp. 1–7

[19] B.A. Hossain and R. Schwitter, "Specifying conceptual models using restricted natural language," in Proceedings of the Australasian Language Technology Association Workshop, 2018, pp. 44–52

[20] J. Mu and A. Sarkar, "Do we need natural language? Exploring restricted language interfaces for complex domains," in Proceedings of Extended Abstracts of the Conference on Human Factors in Computing Systems, 2019, pp. 1–6

[21] M.M Berg, A. Isard, J.D.Moore, "An OpenCCG-based approach to question generation from concepts," in Proceedings of International Conference on Application of Natural Language to Information Systems, 2013, pp. 38–52

[22] J. Schulman et al, "ChatGPT: Optimizing language models for dialogue," 2022. [Online].Available: https://blog.cloudhq.net/openais-chatgpt-optimizing-language-models-for-dialogue/

[23] Y. Wang et al, "Self-instruct: Aligning language model with self generated instructions," 2022, arXiv:2212.10560

[24] F. Petroni et al,"Language Models as Knowledge Bases?," 2019, arXiv:1909.01066

[25] D.k. Misra, J. Sung, K. Lee and A. Saxena, "Tell me dave: Context-sensitive grounding of natural language to manipulation instructions," The International Journal of Robotics Research, vol. 35, no. 1–3, pp. 281–300, 2016

[26] A.Radford et al, "Learning transferable visual models from natural language supervision," in Proceedings of International conference on machine learning, 2021, pp. 8748–8763

[27] R. Girdhar et al, "Imagebind: One embedding space to bind them all," in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2023, pp. 15180–15190

[28] Z. Yang, A. Ishay and J. Lee, "Coupling Large Language Models with Logic Programming for Robust and General Reasoning from Text,", 2023, arXiv:2307.07696