



Learning Hierarchical Robot Skills Represented by Behavior Trees from Natural Language

Kaiyi Wang^{1,2}, Yongjia Zhao^{1,2(✉)}, Shuling Dai^{1,2}, Minghao Yang³,
Yichen He⁴, and Ning Zhang^{1,2}

¹ State Key Laboratory of Virtual Reality Technology and Systems,
Beihang University, Beijing, China
zhaoyongjia@buaa.edu.cn

² Jiangxi Research Institute, Beihang University, Jiangxi, China

³ Institute of Automation, Chinese Academy of Sciences, Beijing, China

⁴ Institute of Intelligent Information Processing, Beihang University, Beijing, China

Abstract. Learning from natural language is a programming-free and user friendly teaching method that allows users without programming knowledge or demonstration capabilities to instruct robots, which has great value in industry and daily life. The manipulation skills of robots are often hierarchical skills composed of low-level primitive skills, so they can be conveniently represented by behavior trees (BTs). Based on this idea, we propose NL2BT, a framework for generating behavior trees from natural language and controlling robots to complete hierarchical tasks in real time. The framework consists of two language processing stages, an initial behavior tree library composed of primitive skill subtrees, and a BT-Generation algorithm. To validate the effectiveness of NL2BT, we use it to build a Chinese natural language system for instructing robots in performing 3C assembly tasks, which is a significant application of Industry 4.0. We also discuss the positive impact of real-time teaching, visual student models, and the synonymous skill module in the framework. In addition to the demonstrated application, NL2BT can be easily migrated to other languages and hierarchical task learning scenarios.

Keywords: Embodied Interaction · Learning from Language · Natural Language Programming · Behavior Tree Generation

1 Introduction

In daily life and industrial production, programmers compose primitive skills (pre-programmed lowest-level skills) of robots into hierarchical skills to achieve various complex robot tasks. However, this can be difficult for users who do not have programming knowledge. Inspired by human social education, interactive task learning (ITL) [15] has been proposed to enable users to teach tasks to robots conveniently. The interaction can be accomplished through demonstration or natural language, but the former is problematic for people with motor

disabilities or those who need to teach fine skills. Therefore, in this paper, we focus on teaching robots skills from natural language.

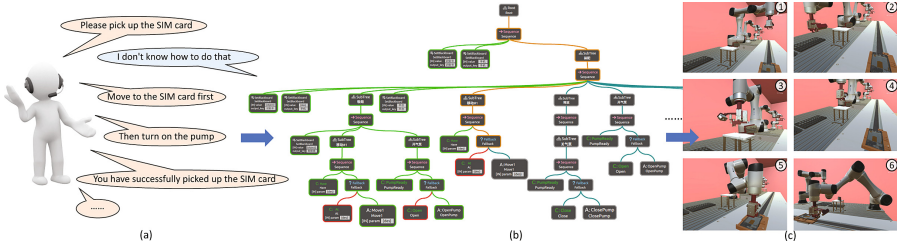


Fig. 1. An example of learning hierarchical skills represented by behavior trees from natural language. (a) The user decomposes high-level skills into low-level primitive skills through dialogue. (b) The learned hierarchical skills are represented and executed through behavior trees. (c) The robot completes manipulation tasks based on the behavior trees. The example given here is transferring a SIM card from the material box to the platform, then to the phone, and finally inserting it into the slot.

An application of learning from language is for assembly robots in Industry 4.0, such as the 3C (computer, communication, and consumer electronics) assembly robots. With the dramatically increasing demand for 3C products, it is an inevitable trend to use robots to assemble them automatically and intelligently. Many skilled workers have little knowledge of robot programming, but they are proficient in the assembly process, so it is a promising way for them to teach assembly tasks using language. The agents are regarded as students with primitive skills, and with step-by-step instructions from human teachers, they can organize primitive skills into hierarchical high-level manipulation skills.

The existing language-based teaching methods for robot manipulation tasks can be divided into two categories: one is based on end-to-end models, and the other is based on interactive task learning. The former focuses on how to perform a task better, without establishing a hierarchical composition structure of the task, which leads to large training loads and low interpretability. Our idea is the latter, which teaches robot tasks through one-shot learning. Despite the impressive results achieved by existing ITL works, there is still a lack of a universal and portable method for representing and generating hierarchical manipulation skills. To address the above issues, we propose NL2BT, a framework for learning behavior trees (BTs) from language and using them to control robots to perform hierarchical skills (see Fig. 1). The main contributions are as follows:

1. An NL2BT framework which enables robots to learn hierarchical skills represented by behavior trees from natural language. It allows users without programming knowledge or demonstration abilities to instruct robots.
2. A generic primitive skill subtree structure that includes execution condition, skip condition and action to ensure efficient execution and logical correctness.
3. A BT-Generation algorithm that generates BTs from semantic information and uses “Blackboard” to achieve parameter mapping for hierarchical skills.

4. A system for learning 3C assembly tasks from Chinese language that validates the framework.

The remainder of the paper is structured as follows: The related work is introduced in Sect. 2. In Sect. 3, we describe the details of the proposed framework. In Sect. 4, we conduct extensive experiments to validate the effectiveness of the NL2BT framework and discuss the impacts of important components. Finally, we conclude the work and outline directions for future work in Sect. 5.

2 Related Work

2.1 Language-Based Imitation Learning

In recent years, many works focus on language-based imitation learning for robot manipulation skills. Stepputtis et al. [25] propose a model for language-based control of articulated robotic arms, Mees et al. [21] provide a public benchmark for instruction following robots, and Google Robotics also proposes the Language Table [20]. They all combine language, vision, and motor to train end-to-end models for robot control, and achieve inspiring results. This shows us a promising future for language-driven robots. However, the end-to-end model is more suitable for learning primitive skills. If used for hierarchical skills, it can result in large data collection and labeling workloads, large training costs, poor interpretability, low reliability, and user customization failure. In contrast, ITL can solve these problems and is more suitable for learning hierarchical skills.

2.2 Interactive Task Learning

ITL is an emerging research topic and its ideas can be seen in many works. She et al. [24] propose a framework for robots to learn high-level actions through dialogue. They adopt combinatory categorial grammar for language processing, and propose a three-layer action representation to execute robot movements. Chai et al. [3] use the tree structure to represent the grounded task. Petit and Demiris [23] use instructions to teach a robot to perform hierarchical hand actions. They extract semantic information according to predefined templates, learn protoactions by mapping semantic information and joint values, and use them to compose high-level actions. ITL can also be applied to software agents. SUGILITE [16, 17] combines instructions with demonstrations to build a conversational assistant on Android. They use a grammar-based executable semantic parser [18] for language understanding and perform tasks with scripts generated by recorded actions. Although these works have achieved impressive results, they have not proposed a general representation and generation method for hierarchical skills that can be widely used and easily ported in ITL systems. Besides, pre-trained language models can be utilized to achieve more robust language understandings. We also notice the existing works that use large language models (LLM) to implement robot tasks, such as ChatGPT for Robotics [26] and SayCan [1]. However, unlike their works, we focus on enabling humans to serve as teachers, so that the robot's behavior fully follows step-by-step instructions without relying on the reasoning of the agent itself, thereby ensuring interpretability.

2.3 Behavior Trees in Robot Manipulation

Finite state machines (FSMs) [8], And-Or graphs [19], semantic graphs [28], and behavior trees [7] are used to represent tasks in robotic manipulation. Among them, BTs have received increasing attention [5]. They are similar to FSMs, but offer advantages in terms of readability, modularity, and real-time performance, making them superior to FSMs for practical applications [6]. Colledanchise and Natale [5] provide a case example of using behavior trees to implement robotic tasks, illustrating the effectiveness of BTs in the representation and execution of robot manipulation skills. French et al. [11] propose an algorithm to learn BTs from demonstrations and validate it with a household cleaning task. However, learning from demonstration can be problematic for users with motor impairment or who need to teach fine tasks. Cao et al. [2] use LLM to generate behavior trees, but do not represent them as executable forms such as XML files or code, so they cannot be directly used to perform actual robot tasks.

3 Approach

3.1 System Overview

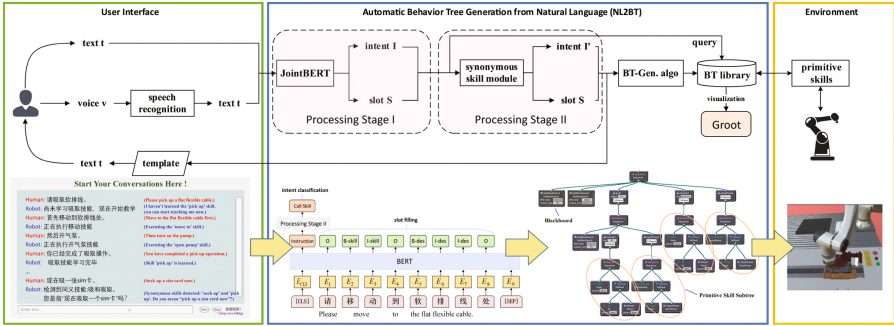


Fig. 2. Architecture of the NL2BT framework and the system built based on it.

As shown in Fig. 2, we propose an NL2BT framework and build a system for learning 3C assembly tasks from Chinese natural language based on it. The user interface (UI) allows users to provide natural language inputs and get agent’s replies. Both voice and text inputs are allowed and the iFLYTEK Open Platform [13] is used for speech recognition. The agent’s reply is generated based on processed semantic information (I, S) and predefined reply templates. In the framework, the text t goes through two processing stages. In the first stage, a JointBERT model [4] is used to obtain the preliminary intent I and slots S . In the second stage, the existing BT library is queried to check if the skill is already learned or has a possible synonymous skill in the library to obtain the final intent

I' . Then (I', S) is used as input of BT-Generation algorithm to generate behavior trees, and the updated BTs can be visualized and executed in real time. Finally, the nodes of the running behavior tree communicate with the robot to perform the skills. In addition to the demonstrated application, NL2BT can be easily migrated to other languages and scenarios (details are given in Sect. 5). Next, we will introduce the implementation of the framework.

3.2 Natural Language Processing

As shown in Fig. 3, JointBERT [4] is leveraged for language processing. We collect 8015 utterances used in the 3C assembly tasks and annotate their intents and slots in BIO format. We use a multilingual BERT [9] model as the pre-trained model and fine-tune it with the labeled data.

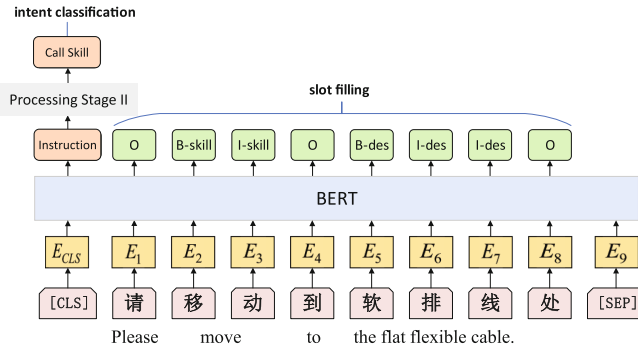


Fig. 3. Intent classification and slot filling for natural language understanding.

The classes of slot S and intent I used are shown in Fig. 4. Intents I are divided into Skill Instruction (“move to the flexible cable”), Completion (“you have successfully assembled the cable”), and Positive/Negative Answer (“yes” or “no”). The slot categories extract important semantic information such as skill names, target objects, destinations, and directions from the utterances.

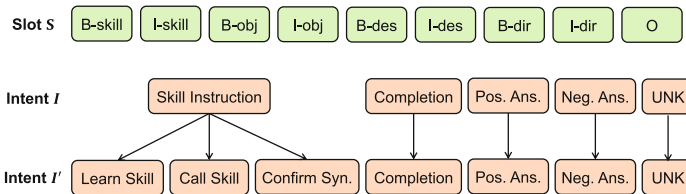


Fig. 4. Classes of slots and intents used in our system.

With the fine-tuned JointBERT model in Processing Stage I, a preliminary intent and slots are given. However, a secondary processing stage is still required before generating behavior trees. When a user inputs a skill instruction, the agent queries the BT library. If it is a learned skill existing in the library, it can be directly executed, so the intent I' is subdivided into “Call Skill”; otherwise, the agent should request instructions from humans, which means “Learn Skill”.

In addition, due to the diversity of language expressions, different skill names may correspond to the same skill composition, which we call “synonymous skills”. For example, “pick up a box” and “grab a box” may have the same decomposed primitive skills. To avoid redundant and repetitive teaching, when the agent receives an unlearned skill name, it compares it with all skill names in the BT library. If a synonymous skill is found, the intent I' is set to “Confirm Synonym”, and the agent will ask the user to confirm whether it is a real synonymous skill. To find synonymous skills, a synonymous skill module is designed. A common idea to capture the similarity between words is to calculate the cosine similarity between their word embeddings. In [27], they provide a Chinese *Synonyms* toolkit based on Word2Vec [22] similarity. However, in addition to the words themselves, we believe that contextual information can also be helpful in synonym judgment, so we use BERT to obtain word embeddings. According to [14], phrase-level information is captured mostly in the lower layers of BERT and gets diluted in the higher layers. Besides, the lower layers capture surface features, which is helpful for judging synonymous skills, especially in Chinese, because the skill names containing the same characters are likely to have similar meanings.

The specific method for judging synonymous skills is shown in Fig. 5. The input is an instruction t_1 containing an unlearned skill name and a constructed comparison instruction t_2 . The construction method of t_2 is to replace the unlearned skill in t_1 with each skill name in the BT library (candidate skill). R_A and R_B are word embeddings of the unlearned skill name and candidate skill name respectively. The cosine similarity between them is calculated as follows.

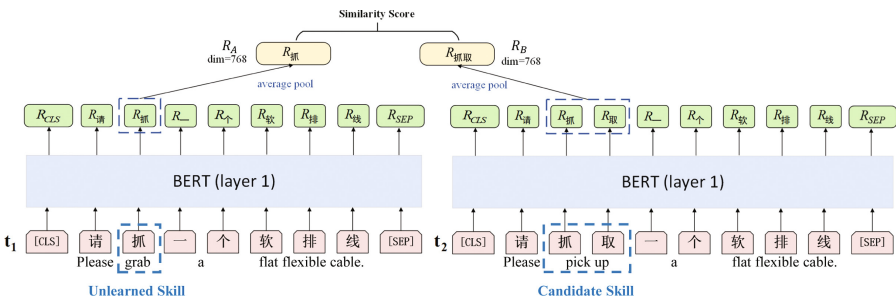


Fig. 5. Method for finding synonymous skills. t_1 is the input instruction, t_2 is a constructed instruction that replaces the unlearned skill with candidate skills from the BT library.

$$similarity\ score = \theta < R_A, R_B > = \frac{(R_A \cdot R_B)}{\|R_A\| \|R_B\|} \tag{1}$$

We construct 200 pairs of data to test the synonymous skill module. As shown in Fig. 6, the best F1 scores using BERT are all higher than that using Word2Vec. Moreover, when using the first layer of BERT, high F1 scores can be obtained over a large threshold range, indicating that the cosine similarity difference between synonymous and non-synonymous skills is large, so the results are more reliable. According to the result, the first layer of BERT and a threshold of 0.465 are selected for the synonymous skill module in our system.

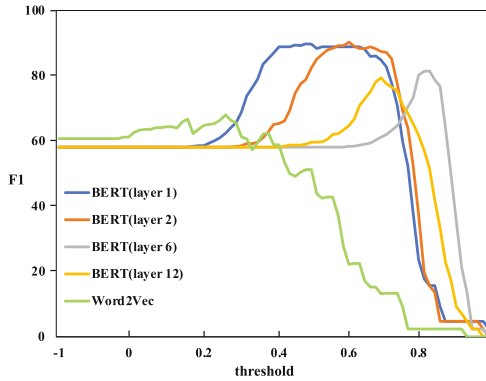


Fig. 6. F1 scores for synonymous skill judgment of each model.

After the above two processing stages, the slots S and intent I' are obtained for the agent’s reply (templates are shown in Table 1) and BT generation.

Table 1. The agent reply templates based on the intent I' and slots S .

Intent I'	Reply Template
Learn Skill	I haven’t learned the [B-skill] [I-skill] skill, you can start teaching me now
Call Skill	Executing the [B-skill] [I-skill] skill.
Completion	[B-skill] [I-skill] is learned.
Confirm Synonym	Synonymous skills detected: [B-skill] [I-skill] and {candidate}. Do you mean ...?
Answer (pos & neg)	OK, ...
UNK	Sorry, I don’t understand.

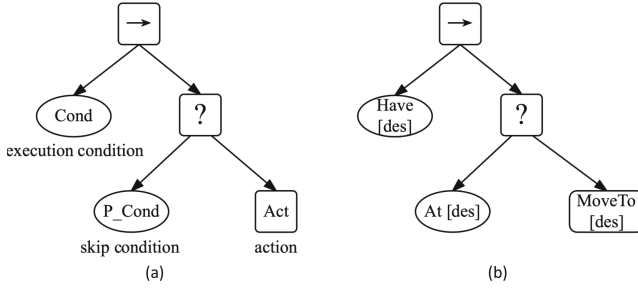


Fig. 7. (a)Structure of primitive skill subtrees. (b)An example of the primitive skill.

3.3 Primitive Skill Subtree

Complex hierarchical skills are composed of primitive skills. In our 3C assembly tasks, the primitive skills include two types of MoveTo (MoveTo₁ [destination] and MoveTo₂ [direction]), OpenPump, and ClosePump, where MoveTo₁ includes a rotation process to align with the object. To compose behavior trees efficiently, we propose a generic structure of primitive skill subtrees. Four standard BT nodes used in it are shown in Table 2.

Table 2. Standard node types used in the primitive skill subtree.





Node Type	Symbol	Description
Sequence		Route ticks to its children from left to right until anyone returns Failure.
Fallback		Route ticks to its children from left to right until anyone returns Success.
Condition		Check a proposition when it receives ticks.
Action		Execute a command when it receives ticks.

Figure 7(a) shows the generic structure of the primitive skill subtree, which serves as the leaf nodes of the generated behavior trees. Each primitive skill has its action, execution condition, and skip condition. The execution condition refers to the condition that must be met to execute the action. As for the skip condition, when it is met, the following action node will be skipped. An example of the primitive skill “MoveTo₁ [des]” is given in Fig. 7(b). When ticks are sent to its Sequence node, it checks whether the destination [des] exists in the environment. If not, the Sequence node stops sending ticks to the Fallback node and the move action cannot be executed. Otherwise, it continues to check whether the robot end effector is already at [des]. If not, the end effector moves to [des],

otherwise, the move action does not need to be executed. The primitive skill subtree structure ensures correct execution logic, avoids system failures with execution conditions, and improves efficiency with skip conditions. Primitive skills used in 3C assembly tasks are shown in Table 3 and the initial behavior tree library is composed of these primitive skill subtrees stored in XML format.

Table 3. Primitive skills used in 3C assembly tasks.

Action	Parameter	Execution Cond.	Skip Cond.
MoveTo ₁	destination	Have [des]	At [des]
MoveTo ₂	direction	/	/
OpenPump	/	Pump Ready	AlreadyOpen
ClosePump	/	Pump Ready	AlreadyClose

3.4 Behavior Tree Generation

The slots S and intent I' are obtained in Sect. 3.2, and an initial BT library with primitive skill subtrees is built in Sect. 3.3. Then hierarchical skills represented by behavior trees can be generated based on them. Despite the 7 intents I' given in Fig. 4, only the intents of Learn Skill, Call Skill, and Completion will be used to generate the behavior trees. We illustrate this in detail in Fig. 8. A BT-Generation algorithm (Algorithm 1) is proposed to generate executable behavior trees from processed semantic information.

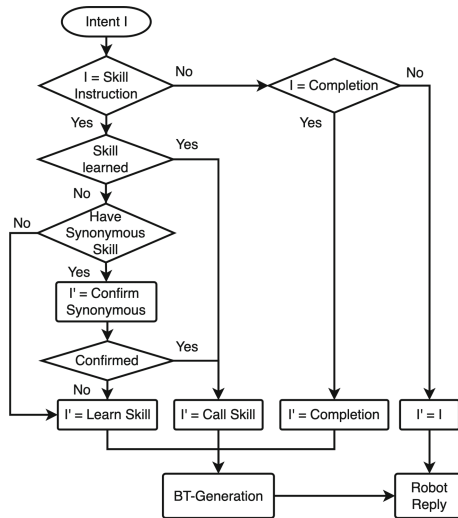


Fig. 8. Illustration of the Processing Stage II.

Algorithm 1. BT-Generation.

Require: *tree, intent, skill, params***Ensure:** *tree*

```

1: function MAIN(tree, intent, skill, params)
2:   Stack.init()
3:   root  $\leftarrow$  tree.getroot()
4:   main_seq  $\leftarrow$  root.find('MainBT').find('Sequence')
5:   while input do
6:     GENERATETREE(intent, skill, params, root, main_seq)
7:   end while
8: end function
9:
10: function GENERATETREE(intent, skill, params, root, main_seq)
11:   if intent = teach_new_skill then
12:     new_tree  $\leftarrow$  root.AddSub("BT", "ID= skill")
13:     seq_node  $\leftarrow$  new_tree.AddSub("Sequence")
14:     for param  $\in$  params do
15:       seq_node.AddSub("SetBlackboard",
16:         "key = param", "value = {param.type}")
17:     end for
18:     if Stack.is_empty() then
19:       grand_seq  $\leftarrow$  main_seq
20:     else
21:       grand_seq  $\leftarrow$  Stack.peek()
22:     end if
23:     ADDELEMENT(grand_seq, params)
24:     Stack.push(seq_node)
25:   else if intent = call_existing_skill then
26:     if Stack.is_empty() then
27:       seq_node  $\leftarrow$  main_seq
28:     else
29:       seq_node  $\leftarrow$  Stack.peek()
30:     end if
31:     ADDELEMENT(seq_node, params)
32:   else if intent = complete_teaching then
33:     Stack.pop()
34:   end if
35:   tree.write()
36: end function
37:
38: function ADDELEMENT(seq_n, params)
39:   for p  $\in$  params do
40:     if p not in seq_n.findall("SetBlackboard").key then
41:       seq_n.AddSub("SetBlackboard", "key = p", "value = p")
42:     end if
43:   end for
44:   seq_n.AddSub('SubTree', 'ID = skill_name', params)
45: end function

```

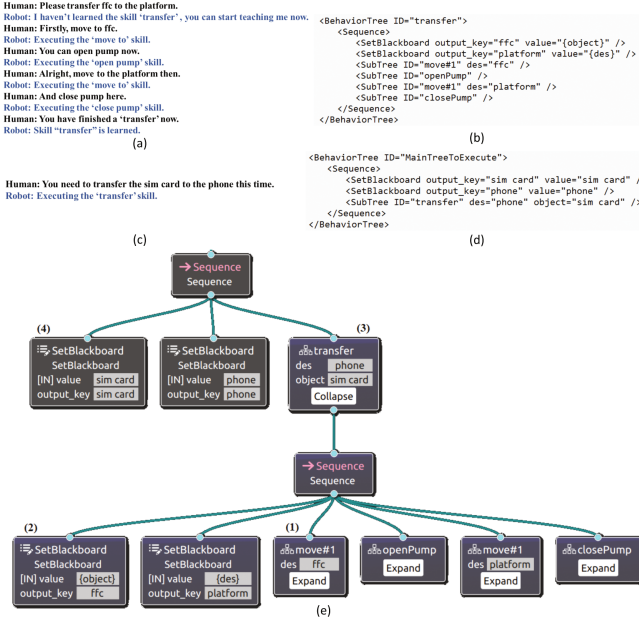


Fig. 9. A case study of parameter generalization. (a) Dialogue for teaching a “transfer” skill; (b) A “transfer” subtree generated from the first teaching in XML format; (c) Dialogue for calling the existing “transfer” skill; (d) BT for performing the “transfer” skill; (e) BT visualization, where (1)–(4) explains the parameter passing in Eq. (2).

Algorithm 1 illustrates how to update the BT library based on different intents. “AddSub” is a factory function used to create a child element of a certain element. Its first parameter is a tag, indicating the type of data being created, and the remaining parameters are attributes represented as key-value pairs. It is worth noting that, as shown in lines 15 and 41, we add “Blackboard”, which implements port remapping between the subtree and the main tree, with different “key”s and “value”s to record input *params*. In this way, the learned skills allow parameter generalization. The “transfer” skill is taken as an example to illustrate this feature. In Fig. 9(a), a user teaches the robot a “transfer” skill, the object of which is a flat flexible cable (FFC), and the destination is a platform. The “transfer” subtree generated from the dialogue is shown in Fig. 9(b), where “ffc” and “platform” are stored as Blackboard keys instead of constant parameters. Their values are the variables that {*object*} and {*des*} denote when the skill is called later. When the “transfer” skill is used again, as shown in Fig. 9(c), its object and destination change into a SIM card and a phone. These two parameters are recorded as the values of “object” and “des”, see Fig. 9(d). As can be seen from Fig. 9(e), the parameter of the first primitive skill subtree *move#1* is “des = ffc” from the first teaching, but the argument for execution is

$$\text{des} \stackrel{(1)}{=} \text{ffc} \stackrel{(2)}{=} \{\text{object}\} \stackrel{(3)}{=} \text{sim card} \stackrel{(4)}{=} \text{sim card (its location)} \quad (2)$$

In this way, “transfer the SIM card to the phone” is decomposed into “move to the SIM card”, “open the pump”, “move to the phone”, and “close the pump”. In other words, when the agent learns how to “transfer A to B”, it no longer needs step-by-step instructions for “transfer C to D”. The BT-Generation algorithm implements the generalization of parameters.

To summarize NL2BT, the agent processes the natural language input in two stages and then updates the behavior trees in XML format in real time. After each update, BTs are visualized using Groot [10] to ensure that the human teachers can view the tree structures and running status, which we call the “student model”, as shown in Fig. 10.

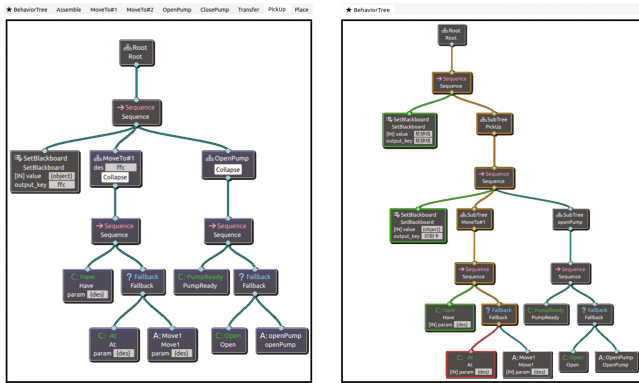


Fig. 10. Using Groot for visual student models in NL2BT. Users can view the existing BT library (left) and the running main tree (right). Colors of the main tree nodes indicate the execution status, orange for Running, green for Success, red for Failure. (Color figure online)

4 Experiment and Result

To validate NL2BT, we build a system for learning 3C assembly tasks from Chinese natural language based on it. We recruit 10 volunteers and introduce them to the functions, usage, and some considerations of the system. Volunteers use our system to teach 3 different 3C assembly tasks to the robotic arm in the virtual environment. Each volunteer teaches in the order of front camera assembly task (Task 1), FFC assembly task (Task 2), and SIM card assembly task (Task 3), which gradually increase in complexity. In Task 1, the robotic arm simply sucks the front camera from the material box and places it in the front camera slot of the phone. In Task 2, the robotic arm first sucks the FFC from the box and releases it to the platform, and then transfers it from the platform to the assembly position of the phone. In Task 3, the robotic arm also transfers the SIM card to the platform first, then to the SIM card slot of the phone, and it also needs to push the card into the slot with the pushing board attached to

the manipulator end. Two experiments are conducted. In the first experiment (see Sect. 4.1), we set up three different teaching modes to analyse the effects of real-time teaching and visual student models. In addition, we also validate the usefulness of the synonymous skill module in the second experiment (see Sect. 4.2) and conduct subjective surveys on all volunteers (see Sect. 4.3).

4.1 Impact of Real-Time Teaching and Visual Student Model

Five volunteers are asked to teach assembly tasks using three different teaching modes shown in Table 4. In the same teaching mode, the skills taught in the previous tasks can be used in the subsequent tasks. Real-time teaching means that the user can always observe the assembly environments, and each language instruction controls actions of the robotic arm in real time. None-real-time teaching means that the language dialogue and robot execution are two separate steps, in which users first generate the BT for the complete assembly process using natural language and then use the generated tree to perform the whole task. Teaching with (without) student model means that the existing BT library and the running status of the main tree are (aren't) visualized.

Table 4. Three different teaching modes.

Teaching Mode	Real-time	Student model
Mode 1	✗	✗
Mode 2	✓	✗
Mode 3	✓	✓

The teaching time of 3 different modes is shown in Fig. 11. In order to reduce the impact of users' familiarity, the teaching order of three modes is random for each user. The result shows that the teaching time of Mode 1 is the shortest, because there are fewer context switches between conversations and skill executions. Besides, users take more time to observe the world environment and robot states while real-time teaching. We also find that when the student model is provided in Mode 3, the teaching time is reduced compared with Mode 2, because the user can obtain the current execution status not only from the environment, but also the visualized running BT. In this way, the efficiency of real-time observation is improved. In addition, although Task 3 is more complicated, the average teaching time of it using Mode 2 and 3 is shorter than that of Task 2, because of the use of learned hierarchical skills. We also find that in complicated tasks with more actions, such as Task 2 and 3, the standard deviation of teaching time is large, and users who are good at teaching and using hierarchical skills complete the teaching much more efficiently.

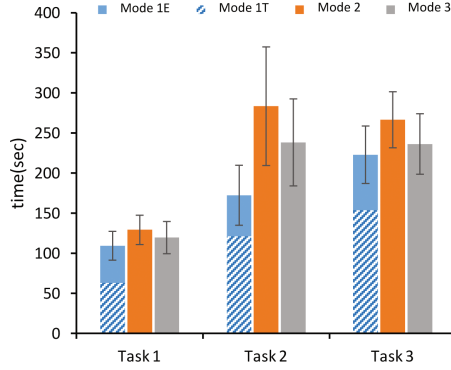


Fig. 11. Average teaching time of 3 teaching modes. Mode 1E refers to the execution time in Mode 1, and Mode 1T refers to the time of human-robot language interaction.

Success rates and numbers of generated hierarchical skills using different teaching modes are shown in Table 5. High success rates demonstrate the effectiveness of NL2BT. Real-time teaching (Modes 2 and 3) has significantly higher success rates, as users can make timely adjustments based on the environment and robot states to ensure successful execution of tasks. Besides, when teaching in real time and being able to visualize the behavior tree library (Mode 3), users prefer to teach and use more hierarchical skills.

Table 5. Success rates and numbers of hierarchical skills of different teaching modes. hier. is short for hierarchical skills.

User	Mode 1		Mode 2		Mode 3	
	success	hier.	success	hier.	success	hier.
1	3/3	5	3/3	4	3/3	5
2	1/3	4	3/3	4	3/3	6
3	2/3	4	3/3	5	3/3	5
4	2/3	4	3/3	5	3/3	5
5	2/3	3	2/3	6	3/3	6
total	66.7%	20	93.3%	24	100%	27

In summary, NL2BT is a feasible and effective framework for learning from natural language. Non-real-time teaching without student models takes less time, but the success rate is lower and fewer hierarchical skills are taught, indicating poor teaching effectiveness. In contrast, real-time teaching takes longer time, but greatly improves the success rate by enabling users to make timely adjustments in a dynamic environment. Furthermore, the student model informs users

of learned skills and running status, allowing them focus more on the agent’s unlearned skills and improve the teaching efficiency. Therefore, real-time teaching and student model are essential for improving the effectiveness and efficiency of learning hierarchical skills using the NL2BT framework.

4.2 Impact of the Synonymous Skill Module

In Sect. 4.1, to better explore the impact of real-time teaching and the visual student model, the BT library is only shared among the same user in the same teaching mode, and the synonymous skill module is excluded. Table 5 shows a total of 71 hierarchical skills are taught. Apart from 45 (5 volunteers * 3 modes * 3 tasks) top-level skills and 10 skills with the same names as others, there are 16 hierarchical skills with only four different composition structures, indicating that many synonymous skills are taught. In this experiment, we add these four skill subtrees to the initial BT library and invite 5 new volunteers to teach 3 tasks using the Mode 3, with and without the synonymous skill module. The average teaching time and the number of stored subtrees are shown in Table 6.

Table 6. Average teaching time and number of stored trees. w/o Syn. denotes teaching without synonymous skill module, and w/ Syn. denotes teaching with it.

	Task	w/o Syn.	w/ Syn.	decreased by
Average Teaching Time	1	111.96	82.66	26.17 %
	2	238.37	202.89	14.88%
	3	300.92	209.24	30.47%
Number of Stored Subtrees	1	11	8	27.27%
	2	14	9	35.71%
	3	16	10	37.5%

Table 6 illustrates that the synonymous skill module helps reduce the teaching time and the number of stored subtrees. In other words, it improves teaching efficiency and maintains a BT library with less storage and memory usage.

4.3 Subjective Feedback

We request that all volunteers evaluate the system’s usability and usefulness using a 7-point Likert scale [12] (ranging from “strongly disagree” to “strongly agree”) based on the statements 1–5 listed in Table 7. Additionally, volunteers participating in Sect. 4.2 are asked to rate an extra statement (statement 6).

We also conduct interviews with volunteers to gather their opinions on real-time teaching, visual student models, and the synonymous skill module. In general, all agree that these features contribute positively to the framework. To be specific, real-time teaching increases users’ confidence in using the system.

Table 7. Average scores on system’s usability and usefulness on a 7-point scale.

Num	Statement	Score
1	I learn to use this system very quickly	6.6
2	The interaction process is simple and easy to understand	6.2
3	All functions in this system are well organized and integrated	5.9
4	I feel confident using the system	6.1
5	Visual behavior tree library is helpful during my teaching process	6.7
6	I find teaching with the synonymous skill module is more efficient	6.4

Additionally, when users couldn’t recall the exact name of a skill that they have taught, they could provide a vague name and rely on the synonymous skill module to suggest candidate skill names. This reduces the need for frequent querying of the BT library and minimizes context switching. One of the participants also mentions that the synonymous skill module and the visual BT library complement each other well. Without the module, the library acts as a teaching manual, requiring users to use the exact skill names to teach efficiently. However, with the module, the BT library becomes a convenient aid for users to determine whether the candidate skills given are true synonymous skills they need. This is helpful for efficient teaching, particularly when numerous skill subtrees are stored in the BT library in the future.

5 Conclusion and Future Work

In this paper, we present NL2BT, a framework for generating behavior trees automatically from natural language and using them to control robots to perform hierarchical tasks. The framework consists of two language processing stages, an initial BT library with primitive skill subtrees, and a BT-Generation algorithm. We develop a Chinese-language system using NL2BT and validate it with 3C assembly tasks in the virtual environment. We also analyse the positive impact of real-time teaching, visual student models, and the synonymous skill module. They improve the success rate and teaching efficiency, reduce memory usage, and receive better user feedback.

While we develop the system for Chinese users, the NL2BT framework can be easily migrated to task learning systems in any language that can be processed by BERT. Furthermore, the framework can be adapted to other scenarios where hierarchical skills are taught using natural language. The system developers only need to fine-tune the language model with their text dataset, find the best hyperparameters (the layer and threshold) for the synonymous skill module, and modify the primitive skills for their own tasks. Therefore, the NL2BT framework has broad application and development prospects.

This paper presents a generic and portable framework for learning hierarchical skills represented by behavior trees from natural language. Future work will

focus on improving and perfecting each part of the framework. For example, natural language generation models can be used to obtain more flexible agent replies, and the best learning methods for primitive skills will be explored. Additionally, the current framework only allows for one intent and skill per input instruction. In future work, we will extend it to support multiple intents and skills using large language models.

Acknowledgments. This research was supported by the National Key Research & Development Program of China (No.2018AAA0102902). We would also like to thank the Institute for Artificial Intelligence, Tsinghua University, for providing equipment and data support.

References

1. Ahn, M., et al.: Do as i can, not as i say: grounding language in robotic affordances. arXiv preprint [arXiv:2204.01691](https://arxiv.org/abs/2204.01691) (2022)
2. Cao, Y., Lee, C.: Robot behavior-tree-based task generation with large language models. arXiv preprint [arXiv:2302.12927](https://arxiv.org/abs/2302.12927) (2023)
3. Chai, J.Y., Gao, Q., She, L., Yang, S., Saba-Sadiya, S., Xu, G.: Language to action: towards interactive task learning with physical agents. In: IJCAI, pp. 2–9 (2018)
4. Chen, Q., Zhuo, Z., Wang, W.: Bert for joint intent classification and slot filling. arXiv preprint [arXiv:1902.10909](https://arxiv.org/abs/1902.10909) (2019)
5. Colledanchise, M., Natale, L.: On the implementation of behavior trees in robotics. *IEEE Rob. Autom. Lett.* **6**(3), 5929–5936 (2021)
6. Colledanchise, M., Ögren, P.: How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Trans. Rob.* **33**(2), 372–389 (2016)
7. Colledanchise, M., Ögren, P.: *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, Boca Raton (2018)
8. De Rossi, G., et al.: Cognitive robotic architecture for semi-autonomous execution of manipulation tasks in a surgical environment. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 7827–7833. IEEE (2019)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint [arXiv:1810.04805](https://arxiv.org/abs/1810.04805) (2018)
10. Faconti, D.: Groot (2018). <https://github.com/BehaviorTree/Groot>
11. French, K., Wu, S., Pan, T., Zhou, Z., Jenkins, O.C.: Learning behavior trees from demonstration. In: 2019 International Conference on Robotics and Automation (ICRA), pp. 7791–7797. IEEE (2019)
12. Hinkin, T.R.: A brief tutorial on the development of measures for use in survey questionnaires. *Organ. Res. Methods* **1**(1), 104–121 (1998)
13. IFLYTEK: Iflytek open platform (2021). <https://www.xfyun.cn>
14. Jawahar, G., Sagot, B., Seddah, D.: What does BERT learn about the structure of language? In: *ACL 2019–57th Annual Meeting of the Association for Computational Linguistics* (2019)
15. Laird, J.E., et al.: Interactive task learning. *IEEE Intell. Syst.* **32**(4), 6–21 (2017)
16. Li, T.J.J., Azaria, A., Myers, B.A.: SUGILITE: creating multimodal smartphone automation by demonstration. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pp. 6038–6049 (2017)

17. Li, T.J.J., Mitchell, T., Myers, B.: Interactive task learning from GUI-grounded natural language instructions and demonstrations. In: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pp. 215–223 (2020)
18. Liang, P.: Learning executable semantic parsers for natural language understanding. *Commun. ACM* **59**(9), 68–76 (2016)
19. Liu, C., et al.: Jointly learning grounded task structures from language instruction and visual demonstration. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 1482–1492 (2016)
20. Lynch, C., et al.: Interactive language: talking to robots in real time. arXiv preprint [arXiv:2210.06407](https://arxiv.org/abs/2210.06407) (2022)
21. Mees, O., Hermann, L., Rosete-Beas, E., Burgard, W.: Calvin: a benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Rob. Autom. Lett.* **7**(3), 7327–7334 (2022)
22. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in Neural Information Processing Systems, vol. 26 (2013)
23. Petit, M., Demiris, Y.: Hierarchical action learning by instruction through interactive grounding of body parts and proto-actions. In: 2016 IEEE International Conference on Robotics and Automation (ICRA), pp. 3375–3382. IEEE (2016)
24. She, L., Yang, S., Cheng, Y., Jia, Y., Chai, J., Xi, N.: Back to the blocks world: learning new actions through situated human-robot dialogue. In: Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL), pp. 89–97 (2014)
25. Stepputtis, S., Campbell, J., Phielipp, M., Lee, S., Baral, C., Ben Amor, H.: Language-conditioned imitation learning for robot manipulation tasks. *Adv. Neural. Inf. Process. Syst.* **33**, 13139–13150 (2020)
26. Vemprala, S., Bonatti, R., Bucker, A., Kapoor, A.: Chatgpt for robotics: design principles and model abilities. *Microsoft Auton. Syst. Robot. Res* **2**, 20 (2023)
27. Wang, H., Hu, Y.: Synonyms (2017). <https://github.com/chatopera/Synonyms>
28. Welschehold, T., Abdo, N., Dornhege, C., Burgard, W.: Combined task and action learning from human demonstrations for mobile manipulation applications. In: 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 4317–4324. IEEE (2019)