ChatBuilder: LLM-assisted Modular Robot Creation

Xin Chen[1], Zherong Pan[2], Xifeng Gao[2], Aiguo Song[1] and Lifeng Zhu[1,*]

Abstract—**Modular robotic structures simplify robot design and manufacturing by using standardized modules, enhancing flexibility and adaptability. However, physical constraints and the need for manual input in design and assembly limit their potential. Current methods to automate this process still require significant human effort and technical expertise. This paper introduces a novel approach that employs Large Language Models (LLMs) as intelligent agents to automate the creation of modular robotic structures. We decompose the modular robot creation task and develop two agents based on LLM to plan and assemble the modular robots from text prompts. By inputting a textual description, users can generate robot designs that are validated in both simulated and real-world environments. This method reduces the need for manual intervention and lowers the technical barrier to creating complex robotic systems.**

I. INTRODUCTION

The development of modular robotic structures has emerged as a powerful approach to simplify the robot design and manufacturing process. By decomposing complex systems into standardized modules, it becomes possible to reduce development cost and enhance the flexibility, scalability, and adaptability of robotic systems. It is essential for modular robots, with their inherent reconfigurability, to be adaptable to different tasks and environments.

However, the modular design approach has limitations. The physical constraints imposed by fixed and standardized modules can restrict the adaptability and versatility of the overall system. For example, the length of robot links can limit the reachability required by a pick-and-place task. The payload of a mobile base can be too small for object transportation tasks. As a result, the design of these standardized modules typically requires manual input, which can be time-consuming and prone to human error. Even more challenging is the task of selecting and assembling these standard modules into a coherent robotic structure — a process that also demands significant manual effort and a high level of technical expertise.

Existing methodologies have sought to address these challenges by using parameterization techniques to streamline the design process and inherently avoid sub-optimal designs, which is a standard procedure in computer-assisted designs [1] . While these approaches offer some degree of automation, they still rely heavily on human intervention to tune the parameters, which possess a considerable technical barrier

for non-experts. Although there are some automated robot design methods, such as optimizing robot design based on evolutionary algorithms [2] or achieving end-to-end robot design through machine learning [3]. Unfortunately, these techniques can be rather computationally and data inefficient.

Recently, LLMs such as ChatGPT have demonstrated remarkable effectiveness in understanding natural language and handling complex tasks. Their success in diverse applications [4] suggests that LLMs could be a powerful tool in automating and simplifying the design process of modular robotic structures, potentially reducing the complexity and technical barriers associated with traditional methods.

In this paper, we propose a novel approach that uses LLMs as intelligent agents to automate the design of modular robotic structures. We divide the general task into two subtasks and design two agents to complete them separately. By simply inputting a textual description of the desired robot structure, users can obtain the corresponding robot design. We demonstrate the generated results and validate the effectiveness of LLM-generated structures in both simulated and real-world environments. Experiments have proven that our method can design effective modular robotic structures, which simplifies the design difficulty of modular robots and lowers the technical threshold, bringing new significance to the automation of modular robot implementation.

II. RELATED WORK

We review related works on LLMs, modular robotics and automated robot design techniques.

a) LLMs: In recent years, LLMs have shown promising application prospects in robotics. Vemprala et al. [5] proposed several principles for applying ChatGPT in robotics. Subsequent work has focused on optimizing the structure of prompts. Singh et al. [6] introduced a procedural LLM prompt structure, enabling the planning function to transcend different environments, robotic capabilities, and tasks. With the integration of Vision Language Models (VLMs), robots can make better plans by considering the current environment [7], [8].

Meanwhile, leveraging GPT-4's powerful multi-modal processing capabilities [9], numerous works have focused on using LLMs as agents to perform complex planning or generation tasks. Zhu et al. [10] proposed KNOWA-GENT, a framework that mitigates planning hallucinations by incorporating external action knowledge into synthesized trajectories. Chain-of-Thought [11] and Chain-of-Code [12] improve model performance on arithmetic, commonsense, and symbolic reasoning tasks. Chen et al. [13] introduced an

*Corresponding author, lfzhulf@gmail.com
[1]Xin Chen, Aiguo Song and Lifeng Zhu are with the State Key Laboratory of Digital Medical Engineering, Jiangsu Key Lab of Robot Sensing and Control, School of Instrument Science and Engineering, Southeast University, China
[2]Zherong Pan and Xifeng Gao are with the LightSpeed Studios Seattle, WA, USA

innovative framework called AutoAgents, which can adaptively generate and coordinate multiple specialized agents based on different tasks. Yuksekgonul et al. [14] introduced TextGrad, a framework that performs automatic "differentiation" through text execution. Liang et al. [15] proposed TaskMatrix.AI, which connects foundational models to millions of APIs to accomplish tasks.

b) Modular Robot: Unlike traditional robotic systems, modular robots are composed of interchangeable and reconfigurable units, allowing for easy customization and adaptation to different tasks. In early research [16], [17], a predefined module library, including joints, links, power units, etc., was used to assemble robots. Similarly, Baca et al. [18] proposed SMART, a method based on three types of modules from a library. These can be assembled in various ways to create different robots for different tasks. Pacheco et al. [19] introduced the Fable system, which consists of self-contained modules equipped with sensors and actuators, allowing users to easily assemble a wide range of robots within seconds. As modular robotics evolved, new types of structures emerged. Lyder et al. [20] introduced the Odin robot, which is based on a deformable lattice and consists of an extendable set of heterogeneous modules. Unlike previous modular robots, they presented the design and implementation of a cubic closed-packed (CCP) joint module, a telescoping link, and a flexible connection mechanism. In summary, these studies primarily focused on the handcrafted modular design of robot structures, while our work intends to automatically adjust the design parameters using LLMs.

c) Automated Robot Design: Previous research has focused on the automatic design of modular robots. The goal of these studies is to find the optimal way to assemble modules from a library to meet the specific needs of a given task. Evolutionary algorithms have been widely used in modular design [2]. Hornby et al. [21], building on evolutionary algorithms and combining them with generative representations, demonstrated an automatic design system that produces complex robots by exploiting the principles of regularity, modularity, hierarchy, and reuse. Zhao et al. [22] proposed RoboGrammar, which generates optimized robot structures to traverse given terrains. They introduced Graph Heuristic Search for efficiently searching in combinatorial design spaces. However, these methods often require multiple iterations of optimization and high computational power, which somewhat limits the application of evolutionary algorithms in modular robot design. Hu et al. [3] presented a novel method based on generative adversarial networks (GANs) that learns a one-to-many mapping from tasks to a distribution of designs. They applied this method to construct locomoting modular robots for terrains with varying obstacle heights and infill. However, this approach heavily relies on the data distribution by itself.

Some research has explored more interactive ways to design robots [23], where users can quickly design robots by dragging graphical symbols. In the method proposed by Lee et al. [24], users can design the initial shape of a robot by sketching, but this approach is still some distance away

from realizing a fully functional robot.

As LLMs continue to demonstrate remarkable results in various fields, some studies have explored the possibility of LLMs guiding robot design [25]. However, this research is mostly conceptual. Similar to our work, Ryan et al. [26] tried to design robots from text descriptions. However, their method only generates a single mesh and does not decompose the mesh into further component assembly instructions.

III. METHOD

Overall, our approach aims to leverage the excellent natural language understanding and code editing capabilities of LLMs. After understanding the user's requirements for a modular robot, the LLM generates code that can model the structure of the modular robot. However, achieving this goal requires considering many issues. We first need a module library to facilitate the LLM's selection from it. Additionally, a complete piece of code for modeling a modular robot is complex and lengthy, making it difficult for the LLM to accomplish the entire modeling task in one go. Inspired by the studies on Chain of Thoughts [11] and Chain of Code [12], we break down the overall task into two subtasks: the planning of modular units and the assembly of those units. Specifically, our ChatBuilder involves two agents: the Planner and the Assembler, as shown in Fig 1. Planner is responsible for the geometric planning of modular units and connection planning, while Assembler is responsible for inferring the local displacement of modular units and executing the assembly. Both agents are based on ChatGPT, and we utilized different prompt engineering techniques to facilitate their distinct functionalities. Finally, ChatBuilder integrates the code generated by both agents and produces a 3D design drawing of the modular robot structure that matches the input text.

A. Module Library

Before all the work, we first need to establish a module library. OpenSCAD [27] was chosen for programming and modeling due to its simple programming language and fast compilation. We designed the module library in OpenSCAD by writing each modular unit as a callable function. This function can generate modular units of different sizes by inputting various parameters. Specifically, our module library includes four types of callable modular units: Link, Joint, Chassis and Wheel, which can change in size and shape according to the parameters. Detailed code for each part in the module library can be found in the supplementary material.

B. Prompt Engineering

For Planner and Assembler, we design two distinct prompts. As shown in Table II, each prompt includes a General Description, a Subtask Description, an Input Example, and an Output Example. The General Description is the same for both prompts, while the Subtask Description is defined based on the specific tasks that each agent needs to accomplish. The Input Example for Planner's prompt is

👤 **User**

Design a six degree of freedom robotic arm structure. I want to use a servo motor with a length of 123mm and a width of 23mm as the drive.
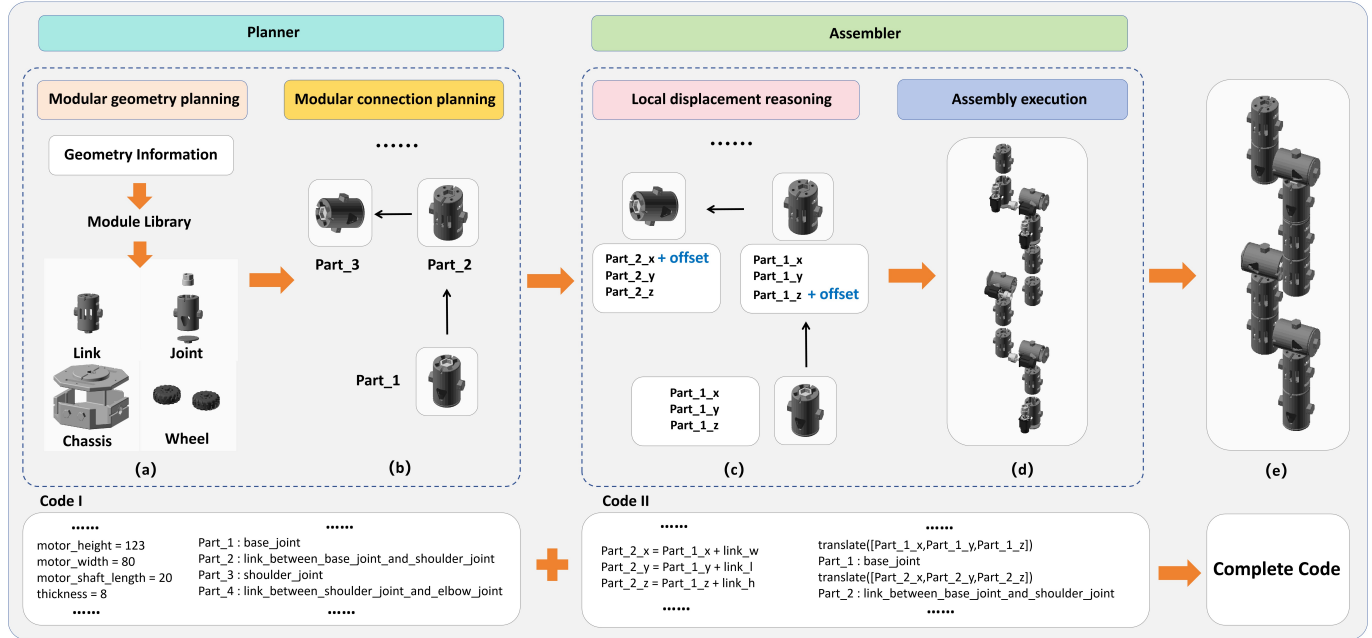
🤖 **ChatBuilder**

Fig. 1. Overview of ChatBuilder. ChatBuilder consists of two agents: Planner and Assembler. The user only needs to input requirements in text form. Planner first retrieves the input text and extracts the geometry information. It then calls the function in module library and inputs the information to generate the modular units. Following this, it plans the topology among all the modules. Assembler, on the other hand, reason the local displacements based on the code output by Planner, issues assembly instructions. Finally, we can get a three-dimensional design drawing of the modular robot that matches the text.

the input text, with the Output Example being the code capable of performing the corresponding parts design tasks. For Assembler, the Input Example is the output code from the parts design agent, and the Output Example is code that performs the corresponding parts assembly.

For the example code, we use the four established modular units to design a six-degree-of-freedom robotic arm structure and a wheeled robot structure with two steering front wheels and two drive rear wheels. Inspired by previous work [6] ,we add extensive comments in the code, which include reasoning guidelines and rules that must be followed (such as rules for selecting modular units and assembling parts) for the agent to reference. The complete prompts can be found in the supplementary material.

C. Planner

The Planner is mainly responsible for understanding the semantics of the input text, and planning the geometry of individual modular units and the topology among all required modular units, and outputting code that performs the corresponding tasks.

a) Modular geometry planning: Our Planner analyzes the input text to extract geometry parameters for modular units. These parameters include motor dimensions (e.g., motor width, motor height, shaft length, shaft radius, and screw hole positions) and custom size information (e.g., wheel radius, chassis length, chassis width, and wall thickness). The Planner then uses these parameters to call predefined modular

unit functions to generate the modular units, as shown in Fig 1 (a). If the input text lacks specific details, our Planner automatically configures appropriate default settings. This functionality allows for flexible modification of the modular unit structure based on partial user requirements.

b) Modular connection planning: After completing the above work, the Planner plans the topology among all modular units one by one. The planner will consider both the overall (input text) and the local (type of the previous part), select the modules that are currently required from the module library, and specify the assembly order of the modules by numbering them, as shown in Fig 1 (b). Specifically, all the modular units have appeared in the Output Example as code comments. The agent only needs to refer to the example to select and plan the parts. This organized numbering system facilitates the assembly process for the Assembler.

D. Assembler

This output code of Planner serves as the input for the Assembler, which is primarily responsible for understanding the topology information contained in the code, deriving the local displacement of each modular unit, providing instructions to transfer each modular unit to the correct position, and outputting code that performs the corresponding tasks.

a) Local displacement reasoning: The Assembler processes the output from the Planner to calculate local displacements, specifically the 3D coordinates of each part. This agent employs a recursive approach to derive these coordinates, where the position of each part is computed
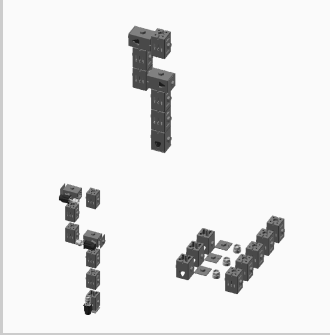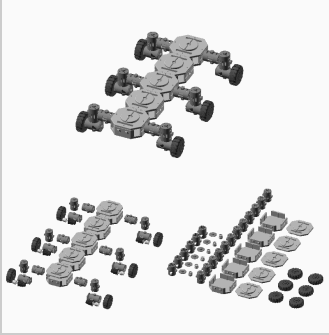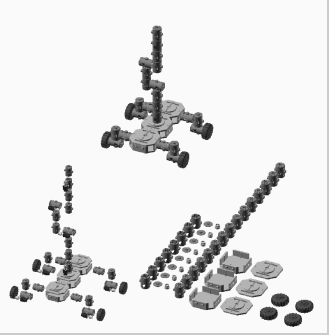
Difficulty	Normal	Normal	Difficult
Input Text	Design a three degree of freedom robotic arm structure. It has a base joint, a shoulder joint and an elbow joint.	Like a four wheeled robot, design a wheeled robot structure with six fixed wheels.	Design a wheeled robot. It has four fixed wheels. And It has a four-degree-of-freedom robotic arm on the body. The arm has a base joint, a shoulder joint, an elbow joint and a wrist joint.
Results			
SucR	100%	100%	80%

TABLE I

GENERATION RESULTS FOR INPUT TEXTS OF DIFFERENT DIFFICULTY LEVELS. SUCR REFERS TO THE SUCCESS RATE.

	Planner	Assembler
General Description	All the same	
Subtask Description	Subtask1 description	Subtask2 description
Input Example	Input text	Parts design code
Output Example	Parts design code	Parts assembly code

TABLE II

THE MAIN COMPONENTS OF THE PROMPTS FOR THE TWO AGENTS.

based on the position of the preceding part plus a specified offset, as shown in Fig 1 (c). The offset can be along multiple coordinate axes or a single coordinate axis.

b) Assembly execution: After determining the local displacements, the Assembler generates assembly instructions by coding the correspondence between each part and its calculated coordinates. These instructions guide the assembly process, ensuring that the parts are correctly assembled into the final structure, as shown in Fig 1 (d). We leverage the semantic understanding capability of the LLM to pair components with their displacements. Specifically, the assembly instructions involve moving each part to its corresponding coordinates. The same operation is performed for each part, and once all parts are moved to their correct positions, an assembly is completed, as shown in Fig 1 (e).

IV. EXPERIMENTS

For the selection of the LLM, preliminary experiments revealed that GPT-4o demonstrates superior performance in both generation quality and time efficiency. Therefore, in the subsequent experiments, GPT-4o will be used as the foundational model. To facilitate subsequent simulation and manufacturing tasks, ChatBuilder outputs three types of 3D design diagrams: an assembly diagram of the modular robotic structure, an exploded view, and a parts list. For simulation validation, we export the modular robotic structure designed by ChatBuilder as STL files and import them into Adams software to verify the function of the generated robot.
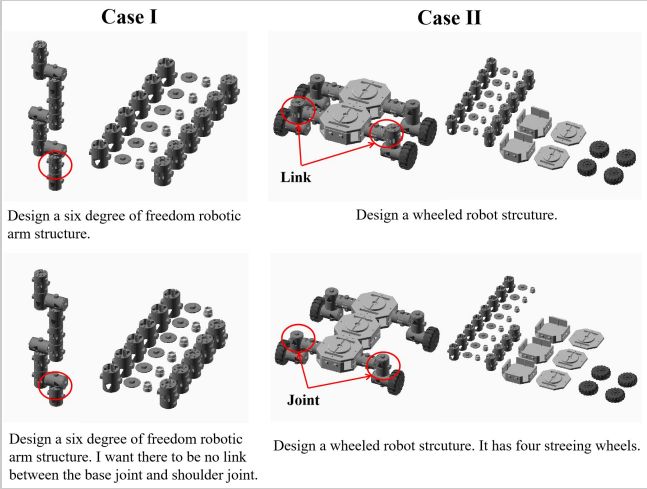

Case I	Case II

Fig. 2. Results of fine-tuning in different cases. For Case I, ChatBuilder can remove a part from a specific location; for Case II, ChatBuilder can replace parts according to requirements.

A. Generation Results

We classify the input text into two levels of difficulty—normal and difficult—based on its similarity to the provided examples. By inputting texts of varying difficulty levels, we obtain a diverse range of generated results. For each input text, we conduct five trials and record the number of times the generated result matched the input requirements. We then calculate the success rate by dividing this count by five. The experimental results are illustrated in Table I. The results demonstrate that for normal level texts, ChatBuilder can accurately design robotic structures that meet the requirements. For difficult level texts, ChatBuilder has a certain probability of failure, but with repeated attempts, it can still design robotic structures that meet the required specifications.
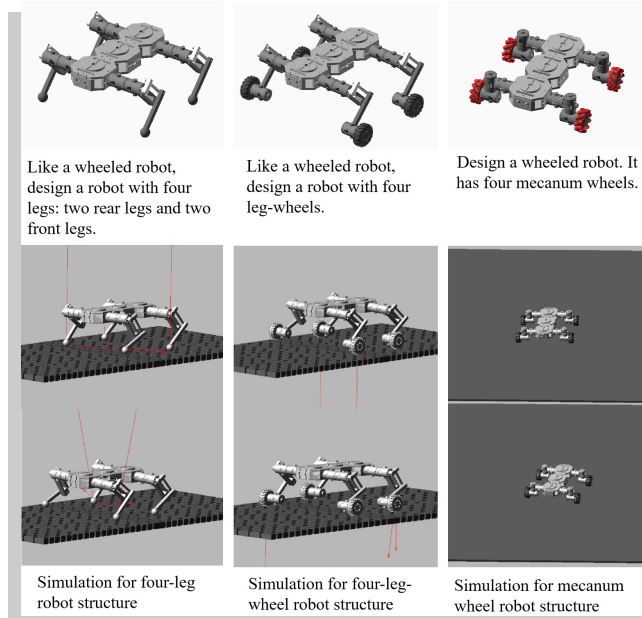
Like a wheeled robot, design a robot with four legs: two rear legs and two front legs.

Like a wheeled robot, design a robot with four leg-wheels.

Design a wheeled robot. It has four mecanum wheels.

Simulation for four-leg robot structure

Simulation for four-leg-wheel robot structure

Simulation for mecanum wheel robot structure

Fig. 3. Generation results and simulation after adding leg, leg-wheel and mecanum wheel components.

Input Text	SucR
Develop a robot equipped with four wheels.	100%
Construct a mobile robot equipped with a set of four wheels.	100%
Produce a robotic system that uses four wheels for locomotion.	80%
Design and create a robot featuring four wheels for movement.	80%

TABLE III

STABILITY VERIFICATION RESULTS OF CHATBUILDER.

B. Fine-tuning Feature

This experiment verifies whether ChatBuilder can understand slight modifications to the input text, thereby enabling more refined designs. For example, adding or modifying requirements on the original design specifications. We test two cases, and the experimental results are shown in Fig 2.

For case I, we want to remove a specific part from the original 6-DOF robotic arm structure. We find that ChatBuilder can accurately identify and remove the corresponding part and make the necessary adjustments to the assembly, resulting in a robot structure that meets the new requirements. For case II, we aim to modify the existing wheeled robot structure by changing all four wheels to steering wheels. We found that ChatBuilder replaced the original link-type parts with joint-type parts at the appropriate locations, transforming the fixed wheels into steering wheels. These cases demonstrate that ChatBuilder can not only design the overall structure of modular robots but also make adjustments to specific details.

C. Parts Expansion

By adding more new components, ChatBuilder can generate a wider variety of modular robotic structures. For this experiment, we add the following new components to the parts library: leg, leg wheel and mecanum wheel. We include descriptions of the new components in the prompt. After completing this, we input some new text, and ChatBuilder

Input Text: Design a three degree of freedom robotic arm structure. It has a base joint, a shoulder joint and a wrist joint. And it has a base underneath it. I want to use a motor with a length of 30mm, a width of 42mm, a shaft length of 22mm and a shaft radius of 5mm.
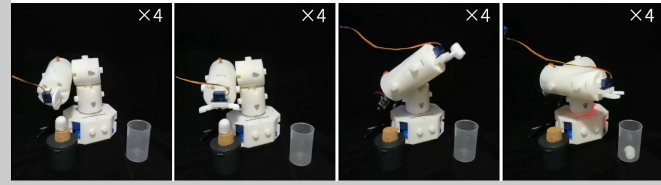

3D Diagram   Real World Robot

Fig. 4. Real-world experimental validation of a three-degree-of-freedom robotic arm structure.

generate modular robotic structures that matched the text descriptions, as shown in the Fig 3. We simulate these structures, and the experimental results show that the structures designed by ChatBuilder are all effective.

D. Stability Verification

We input different text descriptions with the similar meaning and observe the output results to test the stability of ChatBuilder's natural language understanding. We design five different text inputs, all expressing the idea of 'Design a wheeled robot with four wheels.' For each text input, we conduct five attempts, record the number of successful outcomes, and calculate the success rate. The experimental results are shown in the Table III.

E. Real World Experiment

To validate the effectiveness of the modular robotic structures generated by ChatBuilder in the real world, we select a 42 stepper motor as the drive for the real-world robot. This motor has a torque of 0.4 Nm, a length of 30 mm, a width of 42 mm, a shaft length of 22 mm, and a shaft radius of 5 mm. We input the design requirements into ChatBuilder in text form. According to the generated 3D design diagrams, we 3D print the modular unit structures and assemble them to create a real-world modular robot. For 3D printing, we chose the resin material.

ChatBuilder generate a modular robotic structures from text inputs, as shown in Fig 4. We design a simple task with the end effector being a gripper. Task involves using the gripper to pick up an object and place it in a target location. This experiment demonstrate that the modular robotic structures generated by ChatBuilder are effective, manufacturable, and capable of performing simple tasks under reasonable control. The control method used in the experiment is a simple human-in-the-loop control. More complex and precise control can enable the robot to exhibit better performance.

V. DISSCUSIONS

A. Multiple Agents

In our method, the design of agents is a crucial issue. Based on our division of the overall task, the modeling task of a modular robot can be divided into two subtasks: the

	Normal-I	Normal-II	Difficult
ChatBuilder-two agents	100%	100%	80%
ChatBuilder-four agents	100%	20%	0%
ChatBuilder-w/o comments	80%	20%	0%

TABLE IV

CHATBUILDER WITH FOUR AGENTS IS UNABLE TO COMPLETE COMPLEX TASKS. THE PERFORMANCE SIGNIFICANTLY DETERIORATES WITHOUT ADDING COMMENTS IN THE PROMPT.

Design a wheeled robot that can walk on uneven ground.

Fig. 5.   A successful case of task-based input.

planning of individual modular units and the assembly of multiple modular units. The first subtask primarily involves modular geometry and modular connection planning, while the second subtask focuses on local displacement reasoning and assembly execution. Therefore, we attempted to accomplish these tasks using both two agents and four agents. The two-agent approach refers to the Planner and Assembler mentioned in the method section. When further dividing the tasks of the Planner and Assembler, we derive two agents each, resulting in a total of four agents.

We conducted experiments mentioned in the "Generation Result" section using these two approaches. We used text inputs of normal and difficult levels of complexity, performing five trials each to calculate the success rate. The experimental results are shown in the Table IV. The four-agent approach can respond correctly to simple text inputs; each agent only needs to imitate the example to output the correct local code. However, for more complex text inputs, which have a lower degree of similarity to the example in the prompt, each agent focuses only on local code and overlooks the consistency of the overall code. In contrast, the design of the two agents (Planner and Assembler) can coordinate closely related local code, ensuring code consistency.

The prompts for the four agents mentioned in the experiment were simply modified from the prompts of the Planner and Assembler. To achieve better results with the four agents, additional engineering to the prompts are necessary. We also attempted to use a single agent to complete the entire task. However, because the code to generate the modular robot is too long and complex, a single agent was not effective to perform the task limited by the token length of GPT-4o.

B. Comments in code

For the example code in the prompts, we explored the impact of adding comments versus not adding comments on the generation results. Similarly, we input texts of varying difficulty levels for the experiment, and the results are shown in the Table IV. It can be observed that the performance of code generation significantly declines without comments. Our comments contain a substantial amount of reasoning, assembly rules, and other information. Without these details, ChatBuilder can only mimic the example when writing code, making it incapable of performing more advanced reasoning for more complex text inputs.

C. Task-based Inputs

In real world, traditional robot design is usually task-based, while we note that for our approach the input texts
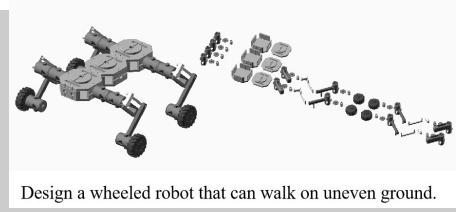
are mainly the description of the robot form but not its function, where the agents are explicitly instructed on the specific structure of the robot to be generated. To verify ChatBuilder's performance with task-based inputs, we also experimented with task-based textual input. For the input "Design a wheeled robot that can walk on uneven ground", ChatBuilder designed a robot with four leg-wheels, as shown in Fig 5, the chassis has a high distance from ground, and leg-wheels are suitable for walking on uneven ground. The result demonstrates that ChatBuilder understood the semantics and made a correct design. However, for the inputs "Design a robot that can run fast", ChatBuilder made some errors in the selection and assembly of parts. The front and rear wheels are not the same height.

The keywords 'wheeled robot' and 'walk' have a high degree of similarity with those in the example, which allows ChatBuilder to find reasoning grounds in the example and make a correct design. However, when only keywords like 'robot' and 'run fast' are input, ChatBuilder cannot find accurate reasoning grounds in the example, leading to situations where the results are partially correct but partially incorrect. More work should be taken to further study the capability of LLM for task-based designs.

VI. CONCLUSIONS

In this paper, we presented a novel approach that leverages LLM as intelligent agents for automating the design of modular robotic structures. The proposed method enables users to generate detailed robot designs from simple textual descriptions, demonstrating effectiveness in both simulated and real-world environments. This approach reduces the manual effort required for design and lowers the technical barriers to creating complex robotic systems.

However, this method comes with certain limitations. Primarily, the focus has been on the structural design of modular robots, with less consideration given to the control aspects of the robots. Although variable modular units allow users to select electronic components (such as motors, batteries, sensors, etc.) based on actual conditions, which enhances the system's flexibility and variability, this also means that the design of the control aspects of modular robots needs to be completed with substantial human supervision. Additionally, the generated designs are largely description-driven, relying on low-level structural details to be provided in the input text. For task-based designs, ChatBuilder can perform some of the work well, but it may require more examples or detailed prompts to improve the accuracy of the designs.

REFERENCES

[1] Machado F, Malpica N, Borromeo S. Parametric CAD modeling for open source scientific hardware: Comparing OpenSCAD and FreeCAD Python scripts[J]. Plos one, 2019, 14(12): e0225795.

[2] R. J. Alattas, S. Patel, and T. M. Sobh. Evolutionary modular robotics: Survey and analysis. Journal of Intelligent and Robotic Systems, 95:815–828, 2019.

[3] Hu J, Whitman J, Travers M, et al. Modular robot design optimization with generative adversarial networks[C]//2022 International Conference on Robotics and Automation (ICRA). IEEE, 2022: 4282-4288.

[4] Xi Z, Chen W, Guo X, et al. The rise and potential of large language model based agents: A survey[J]. arXiv preprint arXiv:2309.07864, 2023.

[5] Vemprala S H, Bonatti R, Bucker A, et al. Chatgpt for robotics: Design principles and model abilities[J]. IEEE Access, 2024.

[6] Singh I, Blukis V, Mousavian A, et al. Progprompt: Generating situated robot task plans using large language models[C]//2023 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2023: 11523-11530.

[7] Huang W, Wang C, Zhang R, et al. Voxposer: Composable 3d value maps for robotic manipulation with language models[J]. arXiv preprint arXiv:2307.05973, 2023.

[8] Zitkovich B, Yu T, Xu S, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control[C]//Conference on Robot Learning. PMLR, 2023: 2165-2183.

[9] Achiam J, Adler S, Agarwal S, et al. Gpt-4 technical report[J]. arXiv preprint arXiv:2303.08774, 2023.

[10] Zhu Y, Qiao S, Ou Y, et al. Knowagent: Knowledge-augmented planning for llm-based agents[J]. arXiv preprint arXiv:2403.03101, 2024.

[11] Wei J, Wang X, Schuurmans D, et al. Chain-of-thought prompting elicits reasoning in large language models[J]. Advances in neural information processing systems, 2022, 35: 24824-24837.

[12] Li C, Liang J, Zeng A, et al. Chain of code: Reasoning with a language model-augmented code emulator[J]. arXiv preprint arXiv:2312.04474, 2023.

[13] Chen G, Dong S, Shu Y, et al. Autoagents: A framework for automatic agent generation[J]. arXiv preprint arXiv:2309.17288, 2023.

[14] Yuksekgonul M, Bianchi F, Boen J, et al. TextGrad: Automatic" Differentiation" via Text[J]. arXiv preprint arXiv:2406.07496, 2024.

[15] Liang Y, Wu C, Song T, et al. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. arXiv 2023[J]. arXiv preprint arXiv:2303.16434.

[16] Farritor S, Dubowsky S, Rutman N, et al. A systems-level modular design approach to field robotics[C]//Proceedings of IEEE International Conference on Robotics and Automation. IEEE, 1996, 4: 2890-2895.

[17] Farritor S, Dubowsky S. On modular design of field robotic systems[J]. Autonomous Robots, 2001, 10: 57-65.

[18] Baca J, Ferre M, Aracil R, et al. A modular robot system design and control motion modes for locomotion and manipulation tasks[C]//2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2010: 529-534.

[19] Pacheco M, Fogh R, Lund H H, et al. Fable II: Design of a modular robot for creative learning[C]//2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015: 6134-6139.

[20] Lyder A, Garcia R F M, Stoy K. Mechanical design of odin, an extendable heterogeneous deformable modular robot[C]//2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. Ieee, 2008: 883-888.

[21] Hornby G S, Lipson H, Pollack J B. Generative representations for the automated design of modular physical robots[J]. IEEE transactions on Robotics and Automation, 2003, 19(4): 703-719.

[22] Zhao A, Xu J, Konaković-Luković M, et al. Robogrammar: graph grammar for terrain-optimized robot design[J]. ACM Transactions on Graphics (TOG), 2020, 39(6): 1-16.

[23] Megaro V, Thomaszewski B, Nitti M, et al. Interactive design of 3d-printable robotic creatures[J]. ACM Transactions on Graphics (TOG), 2015, 34(6): 1-9.

[24] Lee J H, Oh H, Yoon J, et al. RobotSketch: An Interactive Showcase of Superfast Design of Legged Robots[M]//ACM SIGGRAPH 2024 Emerging Technologies. 2024: 1-2.

[25] Stella F, Della Santina C, Hughes J. How can LLMs transform the robotic design process?[J]. Nature machine intelligence, 2023, 5(6): 561-564.

[26] Ringel R P, Charlick Z S, Liu J, et al. Text2Robot: Evolutionary Robot Design from Text Descriptions[J]. arXiv preprint arXiv:2406.19963, 2024.

[27] Gohde J, Kintel M. Programming with OpenSCAD: A Beginner's Guide to Coding 3D-Printable Objects[M]. No Starch Press, 2021.