ChatBuilder: LLM-assisted Modular Robot Creation

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

***Abstract*— Modular robotic structures simplify robot design and manufacturing by using standardized modules, enhancing flexibility and adaptability. However, 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. Most existing work is limited to single design tasks. For example, previous work [1], [2] has optimized mobile robots for different terrains or searched for suitable robotic arms based on end-effector trajectories [3], but it struggles to handle tasks with multiple design requirements simultaneously. Furthermore, while existing work [4] has achieved some level of automation and simplification in the design process, it still necessitates significant manual effort and expertise. This not only increases design time but also restricts the involvement of non-expert users in robot design.

Recently, LLMs have demonstrated remarkable effectiveness in understanding natural language and handling complex tasks. Their success in diverse applications [5]–[7] 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. With only few-shot in-context learning, our agents can understand and analyze input text, select appropriate components, and ultimately assemble modular robots that meet the requirements. 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 barrier for non-experts.

*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 independent researchers at Seattle, WA, USA

Project website: https://fodechain.github.io/ChatBuilder/

II. RELATED WORK

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

a) LLMs: Leveraging LLMs' advanced reasoning and contextual understanding capabilities [8], numerous studies have explored their use as agents for complex planning and generation tasks [5], [9], [10]. Chain-of-Thought [11] and Chain-of-Code [12] improve model performance on arithmetic, commonsense, and symbolic reasoning tasks.

In-context learning is one of the key methods for LLMs as intelligent agents. LLMs perform analogical learning through demos and can still demonstrate good performance without explicit training [13]. Previous work [14] has also found that LLMs are better at learning from code-style demos. Inspired by these works, we attempt to enhance the LLM's ability to design modular robots through in-context learning, using high-quality examples of modular robot designs.

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. Early research [15], [16] utilizes a predefined module library, including joints, links and power units, to assemble robots. Similarly, [17], [18] also introduce a module library. These modules can be assembled in various ways to create different robots for different tasks.

In summary, these studies primarily focus on the handcrafted modular design of robot structures, while our work intends to automatically plan the components of a modular robot 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 [19]. Some methods utilize optimization or machine learning, such as terrain-based optimization of robot structures [1], [2] and end-trajectory-based robot design [3],

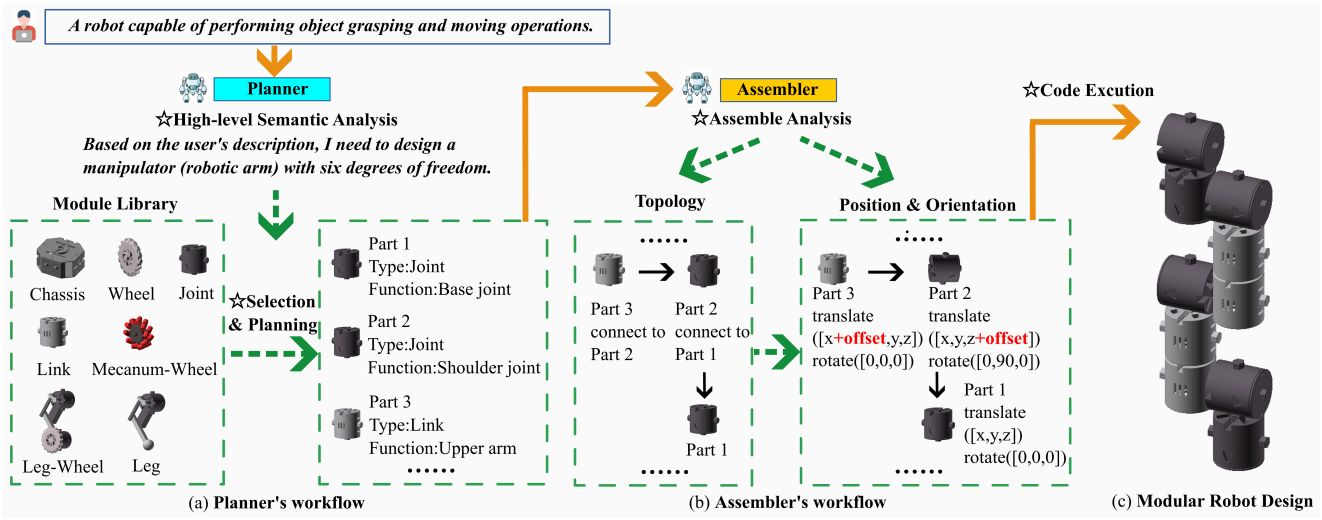


Fig. 1: Overview of ChatBuilder. ChatBuilder comprises two agents: the Planner and the Assembler. The user is required to provide input solely in textual form. The Planner conducts a high-level semantic analysis of the input, identifying the necessary components and their respective functions within the design. The Assembler, based on the Planner's output, analyzes the topological relationships among components, including their position and orientation, and generates standardized code. By executing this code, a modular robot design that meets the specified requirements can be obtained.

[4]. However, these methods often require multiple iterations of optimization and high computational power. In addition, while these approaches demonstrate excellent performance, they tend to focus on a specific task.

Some research has explored more interactive ways to design robots [20], [21], where users can quickly design robots by dragging graphical symbols. In [22], 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. The work in [23] investigate the use of LLMs in design and manufacturing, revealing their potential. Similar to the goal of [20], we also want to find a common way to customize robots quickly and easily. In contrast, we leverage the generality of text to develop a novel text-based interaction, enabling users to design modular robots through simple text input.

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 require a module library to enable the LLM to select from it, where the modular components are described in clear, interpretable text. 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 components and the assembly of those components. Specifically, our proposed method, ChatBuilder, consists of two agents: the Planner and the Assembler, as shown in Fig. 1. The Planner is responsible for performing high-level semantic analysis of the input text, identifying the necessary components, their functions in the design, as shown in Fig 1 (a). The Assembler analyzes the topological relationships between components, along with their position and orientation information, based on the output of the Planner, then outputs standardized code, as shown in Fig 1 (b). By executing the code, we can obtain modular robot design that meet the design requirements, as shown in Fig 1 (c). We will explain our approach in more detail in the following sections.

A. Module Library

Before all the work, we first need to establish a module library. OpenSCAD [24] was chosen for programming and modeling due to its simple programming language and fast compilation. We design the module library in OpenSCAD by writing each modular component as a callable function. Specifically, our module library includes four types of callable modular components: Link, Joint, Chassis, and Wheel, as shown in Fig 1 (a). In addition to these basic components, we incorporate high-level components into the module library, including Leg, Leg-Wheel, and Mecanum-Wheel. This expansion broadens the design space and prevents the Planner's design capability from being constrained by module library limitations, which could lead to design failures. Further details on the module library are available on our website.

B. Planner

The Planner plays a crucial role throughout the design process. It is responsible for analyzing high-level tasks and selecting components based on design requirements, as shown in Fig. 1 (a). First, it is essential to ensure that the Planner's planning occurs within a predefined set of standards and the design space we define. To facilitate this, the Planner is informed of the components in the module library, including their names and dimensions. Since different

Planner's example 1 (manipulator)

Demo Input	
Design requirement	*A robot capable of grasping objects in open spaces.*
Demo Output	
High-level semantic analysis	*A manipulator(robotic arm) with six-degree-of-freedom.*
Structured list of components	*part_1:* *//function: a base for the robotic arm* *//type: chassis* *part_2:* *//function: Base Rotation Joint.* *Allows the robotic arm to rotate around the vertical axis of its base.* *//type: joint* *……*

Fig. 2: Composition of the Planner example. The figure shows a part of one example (a manipulator).

components serve various functions across different tasks and robots, the Planner is provided with examples of component functionalities, specifically through two examples: a manipulator and a mobile robot. Each example includes a Demo Input and a Demo Output. Fig. 2 illustrates part of a Planner example. The Demo Input corresponds to the high-level design requirements. The Demo Output consists of two parts: the first involves analyzing the design requirements, including classifying the required robot, extracting and analyzing key information, and performing some quantitative calculations and reasoning processes; the second part is a structured list of components, which includes the ID of each required component, the function of each component within the specific design task, and the type of each component.

C. Assembler

The Assembler is responsible for converting the Planner's output into assembly code. Specifically, the Assembler analyzes the topological relationships between components and the pose information of each component, as illustrated in Fig. 1 (b). The input to the Assembler corresponds to the output from the Planner. In contrast to the Planner, the Assembler assembles the modular robot according to predefined assembly rules. It provides the pose of each component, which includes the specific coordinates and orientation of each component. A recursive approach is employed to derive these coordinates, where the position of each component is calculated based on the position of the preceding component plus a specified offset. The offset can be along one or more coordinate axes. Each component has multiple connection interfaces, and the method of connecting components depends on the specific task and the role of the component in that task, as extracted from the Planner's output. Similar to the Planner, two assembly examples—a manipulator and a mobile robot—are provided for the Assembler to learn from, as shown in Fig. 3. Each example includes a Demo Input and a Demo Output. The Demo Input for the Assembler corresponds to the Demo Output from the Planner, while the Demo Output consists of the assembly code, including component types and pose analysis, expressed in comments. The complete set of prompts is available on our website.

Assembler's example 1 (manipulator)

Demo Input	
Demo Output of Planner's example	*A manipulator(robotic arm) with six-degree-of-freedom.* *part_1:* *//function: a base for the robotic arm* *//type: chassis* *part_2:* *//function: Base Rotation Joint.* *Allows the robotic arm to rotate around the vertical axis of its base.* *//type: joint* *……*
Demo Output	
Standardized assembly code	*//part_1* *//type: chassis* *//position: origin position (0,0,0)* *//orientation: origin orientation.facing the positive z direction* *part_1_x_t = 0;* *part_1_y_t = 0;* *part_1_z_t = 0;* *translate([part_1_x_t, part_1_y_t, part_1_z_t])* *rotate([0,0,0])* *chassis();* *//part_2* *//type: joint* *//position: The negative z direction of part_2 (joint) is connected to the positive z direction of part_1 (chassis). part_2 (joint) has an offset in the positive z direction relative to part_1 (chassis).* *//orientation: origin orientation. facing the positive z direction* *part_2_x_t = part_1_x_t;* *part_2_y_t = part_1_y_t;* *part_2_z_t = part_1_z_t + chassis_positive_z + joint_negative_z;* *translate([part_2_x_t, part_2_y_t, part_2_z_t])* *rotate([0,0,0])* *joint();* *……*

Fig. 3: Composition of the Assembler example. The figure shows a part of one example (a manipulator).

IV. EXPERIMENTS

Our experiments investigate the following research questions in the subsequent three sections:

Q1: To what extent does ChatBuilder accommodate diverse design requirements?

Q2: How effectively does ChatBuilder accomplish specific tasks?

Q3: Does the text-based interaction approach enhance efficiency and reduce the barriers to robot design?

Previous robot design efforts primarily focus on specific tasks [2], [3], with evaluation methods and metrics varying significantly, making fair baseline comparisons difficult. In contrast, this work addresses a broad range of design requirements rather than a single specialized task. Instead of emphasizing performance on a single task, we assess whether ChatBuilder consistently meets diverse design requirements while aligning with textual semantics. Following the general principles of robot design in [25], we evaluate ChatBuilder-generated designs based on the following criteria:

a) ***Assembly Validity****:* This criterion assesses whether the components are free from interference or overlap and whether they are correctly connected through designated interfaces. We verify Assembly Validity using Adams [26], a multi-body dynamics simulation software, to check whether the assembled modular robot simulates well.

b) ***Requirement Compliance****:* This criterion evaluates whether the designed modular robot adheres to the structural and functional specifications described in the input text. Re-

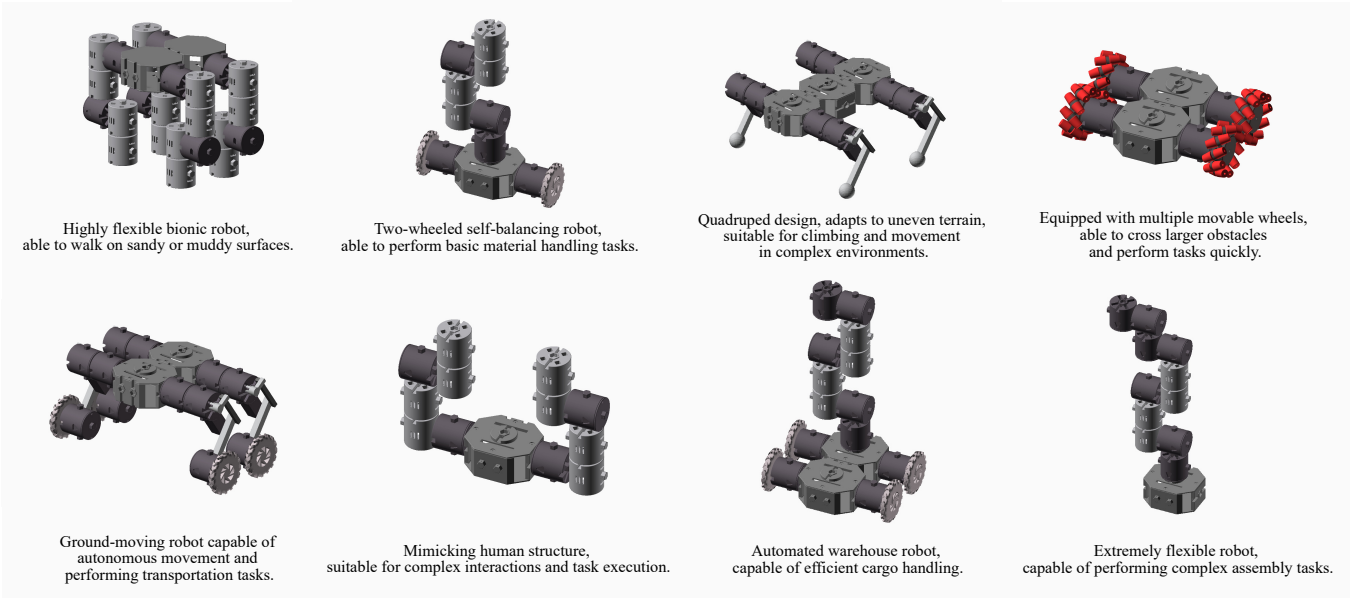


Fig. 4: Qualitative results generated by our approach. Given a textual input, the approach accommodates a wide range of design requirements and produces diverse robot designs that meet the specified criteria.

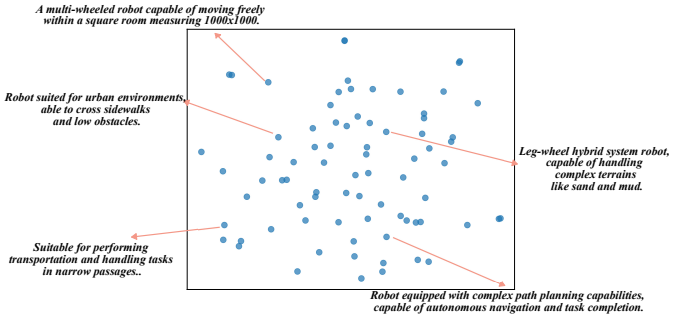


Fig. 5: Visual distribution of test texts. Each point represents a text. The 78 texts are evenly distributed.

quirement Compliance is determined through a combination of simulation and expert judgment. Specifically, we invite multiple experts to evaluate the generated results and collect their scores. The average score is then calculated, and if it exceeds 8 (out of 10), the generated result is considered to meet the design requirements.

Similar to the approach in [25], we collect and analyze various robot design requirements expressed in textual descriptions. We exclude requirements that fall outside our design scope, as well as duplicate or highly similar entries, resulting in a final set of 78 test texts. The complete set of test texts is available on our website. Importantly, rather than focusing on the number of test texts, we prioritize ensuring they encompass a diverse range of design requirements to validate the breadth of ChatBuilder's design capabilities. To visualize this diversity, we use BERT [27] to encode the test texts into vector representations and project them into a two-dimensional space, as shown in Fig. 5. The distribution indicates that the test texts are well spread, demonstrating their broad coverage.

A. Results

Fig. 4 presents the qualitative results generated by our approach. For each input text, we repeat the process 20 times and record the number of instances where ChatBuilder produces assembly-valid designs or meets the specified requirements to calculate the success rate. Additionally, we calculate the cosine similarity between each test text and the Demo Inputs used for in-context learning. We define the cosine similarity as $\min\left(\mathbf{T}\cdot\mathbf{D}_1, \mathbf{T}\cdot\mathbf{D}_2\right)$, where $\mathbf{T}$ represents the unit vector of the test text, and $\mathbf{D}_1$, $\mathbf{D}_2$ represent the unit vectors of the two Demo Inputs of the Planner.

As shown in Fig. 6, when the similarity between the test text and the Demo Input is high, ChatBuilder exhibits a higher success rate in both Assembly Validity and Requirement Compliance. Even when the similarity is low, ChatBuilder can still achieve a high success rate in design completion. However, it is important to note that lower success rates (below 50%) are more likely to occur when the similarity between the test text and the Demo Input is low. Furthermore, we observe that Assembly Validity and Requirement Compliance are consistent in most cases. Nonetheless, when the similarity between the input text and the Demo Input is low, ChatBuilder more frequently generates results that satisfy Assembly Validity but fail to meet Requirement Compliance.

Across all 20 repeated trials, the average number of assembly-valid results is 16, while the average number of results meeting the specified requirements is 15. We adopt Requirement Compliance as the final evaluation criterion. In other words, ChatBuilder generates modular robot designs that satisfy the textual requirements with an average success rate of 75%.

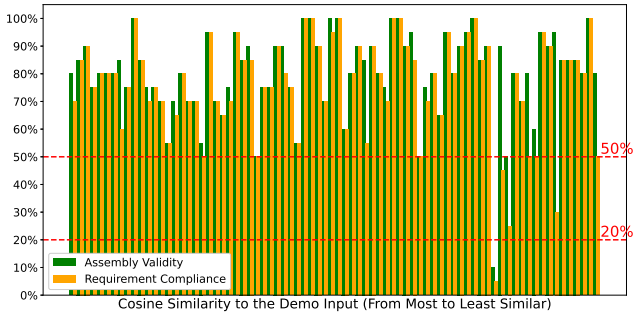
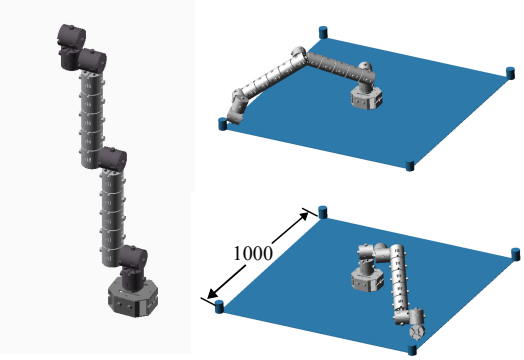


Fig. 6: Quantitative results of ChatBuilder across all 78 test texts. Green bars indicate the success rate of generated results being assembly-valid over 20 trials, while yellow bars represent the success rate of generated results meeting the specified requirements. The results show that lower success rates are more likely to occur when the similarity between the test text and the Demo Input is low.



(a) Modular robot design generated by ChatBuilder.

(b) Simulation of tasks with quantitative constraints.

Fig. 7: Generated result for ***"In a square room measuring 1000x1000, a robotic arm positioned at the center of the room is capable of extending its end effector to any of the four corners of the room."***

B. Case Study

To investigate the design performance of ChatBuilder on specific tasks, we conduct several case studies.

For the input text containing quantitative constraints:

In a square room measuring 1000x1000, a robotic arm positioned at the center of the room is capable of extending its end effector to any of the four corners of the room.

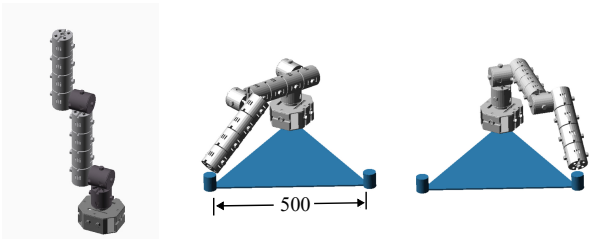
We identify ChatBuilder's reasoning process in the Planner's output:

The diagonal of the square room is approximately 1414 units, requiring an arm length of at least 707 units to extend from the center to any corner. The robotic arm's structure uses multiple links, with an accumulated length greater than or equal to 707 units for both the upper and lower arms combined.

ChatBuilder generates a six-degree-of-freedom robotic arm. Furthermore, its position is shifted from the origin (0, 0, 0) to (500, 500, 0), where the coordinates represent (x, y, z). The number of links composing the upper and lower arms is increased to five. We simulate the generated design, and the robotic arm is indeed able to reach any corner of the square room, as shown in Fig. 7.

Fig. 8 presents the generated results for other input texts

In an equilateral triangular room with each side measuring 500 units, a robotic arm positioned at one corner of the triangle is capable of extending its end effector to reach either of the other two corners of the room.



A multi-wheeled robot capable of moving freely within a square room measuring 1000x1000.

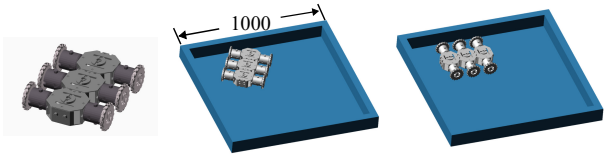


Fig. 8: Generated results and simulation for other input texts containing quantitative constraints.

Simplified wheeled robot design, able to cross small obstacles.

Fig. 9: ChatBuilder demonstrates diversified generated results for the same design requirements. It exhibits both robustness and diversity.

containing quantitative constraints. Based on the dimensions of the components, ChatBuilder can perform basic computational reasoning to fulfill the textual requirements and determine the number of components to select. It is important to note that the Demos used for in-context learning do not include examples of mathematical calculations, meaning these computations are performed in a zero-shot manner.

Additionally, ChatBuilder generates diverse outputs for the same textual requirement while ensuring compliance with the specifications, as illustrated in Fig. 9. This demonstrates its robustness and diversity.

C. User Study

To verify whether the text-based interaction approach in ChatBuilder can enhance design efficiency and lower the design barrier, we conduct a comparative experiment with a graphical user interface (GUI), which is a design approach that allows users to design robots by dragging components.

We invite 10 experts and 10 non-experts to participate, with experts having extensive modeling experience and specialized knowledge in robotics design, while non-experts have no prior experience in modeling or robotics design. Participants are asked to complete the same design task using both interaction methods. For each participant, we randomly select a text from the 78 test texts, and users

	GUI	ChatBuilder	Freq for ChatBuilder
Experts	223s	135s ↓	2
Non-experts	549s	147s ↓	3

TABLE I: Participants' time usage. The first two columns show the average time to complete a design using the GUI and ChatBuilder respectively, while the third column indicates the average number of times ChatBuilder is used. Both experts and non-experts require less time when using ChatBuilder for robot design.

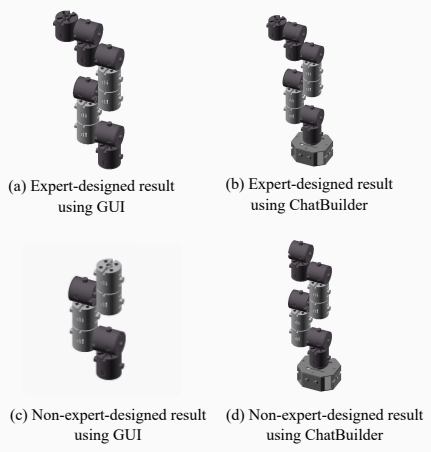


Fig. 10: Different designs obtained using two interaction tools for the same requirement, ***"A flexible robot capable of performing grasping tasks."*** ChatBuilder helps non-experts create more comprehensive and refined designs, while also producing results that are close to those of experts.

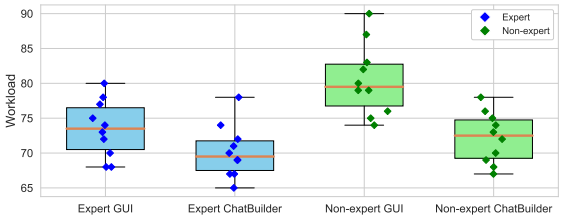


Fig. 11: Participant feedback on the workload of two different interaction methods. Both experts and non-experts report a lower workload when using ChatBuilder for design tasks, with a more significant difference observed among non-experts.

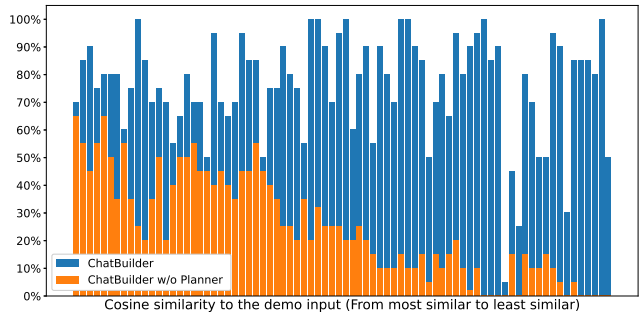


Fig. 12: The absence of a Planner significantly reduces ChatBuilder's success rate in Requirement Compliance, especially when handling more challenging tasks (where the similarity between the input text and the Demo Input is low), to the point where the task can not even be completed.

are allowed to use any component from the module library to design a modular robot that meets the requirements. To ensure fairness, both experts and non-experts receive diagram and code versions of the Demo used for in-context learning, helping them gain a fundamental understanding of the module library and design space. Additionally, we provide training for non-experts on how to use the GUI.

As shown in Table I, both experts and non-experts spend less time using ChatBuilder compared to the GUI, with non-experts showing a significant reduction in design time when using ChatBuilder. The number of times ChatBuilder is used by both non-experts and experts is greater than 1 but less than 3. Even so, the total design time using ChatBuilder remains shorter than that using the GUI. In practical applications, users can repeatedly invoke ChatBuilder within a short time frame to meet their design requirements.

Fig 10 presents one of the design results from different participants using different interaction tools. ChatBuilder helps non-experts create more comprehensive and refined designs, while also producing results that are close to those of experts.

Inspired by NASA-TLX [28], we design a 10-point Likert scale questionnaire to collect feedback from participants regarding the workload of the two interaction approaches. As shown in Fig. 11, both experts and non-experts report a lower workload when using ChatBuilder for design tasks, with this effect being particularly pronounced among non-experts. These results demonstrate that ChatBuilder improves design efficiency and lowers the design barrier for non-experts.

D. Ablation

In the ablation study, we investigate the impact of the Planner on the generated results, as well as the effect of the number of Demos on the generated outcomes.

We still use test texts for the trials, repeating each text 20 times, and calculate the number of results that meet the requirements (Requirement Compliance) to determine the success rate. Additionally, we count the number of times different generated results occur across all repetitions to assess Diversity.

We try using a single agent to directly learn the final assembly code through context, eliminating the Planner. As shown in Fig 12, its overall success rate decrease. This is particularly noticeable for input texts that are less similar to the Demo Input, where the success rate drop significantly. In other words, performing an initial analysis of the input text through the Planner before generating the final code can improve the success rate of ChatBuilder.

We also test the impact of different numbers of Demos on the generated results. Fig 13 shows that when the number of Demos increase from 1 to 2, both the success rate in Requirement Compliance and Diversity improve. However, when the number of Demos exceeds 2, the success rate does not show significant changes. In fact, when the number of Demos reaches 5, the success rate decreases. This is due to the truncation problem [29]: more Demos exceed the input length limit of the LLMs, causing the Demos to be truncated and reducing the success rate. Additionally, we find that when the number of Demos exceeds 2, the Diversity of

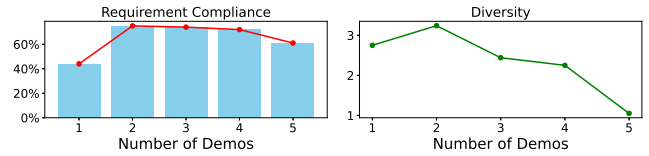


Fig. 13: As the number of Demos increases, the success rate in Requirement Compliance and Diversity initially rises and then decreases.

the generated results decreases, and the results become more monotonous. This is because ChatBuilder tends to directly imitate the existing Demos, which limits its creative ability.

We also compare the impact of using different LLMs on the experimental results. We try GPT-3.5, GPT-4 and GPT-4o. It is found that GPT-4o outperform the other models in both response time and success rate.

E. Real World Experiment

We build some prototypes based on the design diagrams generated by ChatBuilder. We select a stepper motor as the drive for the real-world robot. The motor has a torque of 0.4 N·m, a length of 30 mm, a width of 42 mm, a shaft length of 22 mm, and a shaft radius of 5 mm. We 3D print the modular components and assemble them to create a real-world modular robot. For 3D printing, we choose the resin material.

For the manipulator generated by ChatBuilder, we use the gripper as the end-effector, allowing the manipulator to pick up an object and place it in a target location, as shown in Fig 14 (a). For the mobile robot generated by ChatBuilder, it smoothly navigate around obstacles and pass through narrow passages, as shown in Fig 14 (b). The control method used in the experiment is a human-in-the-loop control. More precise control can enable the robot to exhibit better performance.

V. DISCUSSIONS

In this study, we acknowledge that while research [30] suggests that LLMs are not particularly strong in continuous numerical reasoning, defining modular components clearly helps guide the design process within a discrete design space, enabling more effective solution generation using LLMs.

Regarding evaluation metrics, we recognize that assessing robot design involves multiple factors, including structural soundness, control efficiency, and reliability. In this work, we rely on expert judgment to account for these aspects. The success rates in our preliminary study may be influenced by subjectivity, as evaluation criteria vary across design requirements. To mitigate this, we incorporate multiple experts and take the average of their ratings. This work represents an initial step in exploring LLMs' capabilities in robot design, and we continue to refine both our design and evaluation system.

VI. CONCLUSIONS

In this paper, we present a novel approach that leverages LLM as intelligent agents for automating the design of modular robotic structures. The proposed method enables

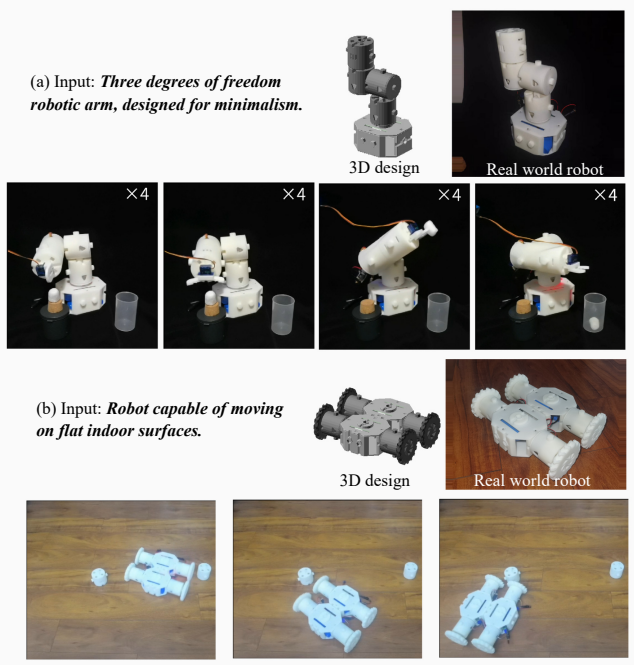


Fig. 14: Real world experiments. We build several prototypes and use them to complete the tasks.

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 has certain limitations. Primarily, the focus has been on the structural design of modular robots, with less attention given to the control aspects. Additionally, although ChatBuilder can modify the robot's structure to meet design requirements, it currently does not account for factors such as energy consumption and power. Furthermore, while ChatBuilder is capable of handling tasks with quantitative constraints, its success rate is lower compared to tasks without such constraints. In the future, we will incorporate additional factors of robot control to further enhance modular robot design. We will also involve more participants to improve the evaluation of our system. While this work primarily addresses the structural design of modular robots, we envision leveraging advancements in generative artificial intelligence to push the boundaries of automated robot design, extending beyond modular systems to more versatile and intelligent robotic solutions.

REFERENCES

[1] J. Hu, J. Whitman, M. Travers, and H. Choset, "Modular robot design optimization with generative adversarial networks," in *2022 International Conference on Robotics and Automation (ICRA)*, pp. 4282–4288, IEEE, 2022.

[2] A. Zhao, J. Xu, M. Konaković-Luković, J. Hughes, A. Spielberg, D. Rus, and W. Matusik, "Robogrammar: graph grammar for terrain-optimized robot design," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 6, pp. 1–16, 2020.

[3] R. Desai, M. Safonova, K. Muelling, and S. Coros, "Automatic design of task-specific robotic arms," *arXiv preprint arXiv:1806.07419*, 2018.

[4] S. Ha, S. Coros, A. Alspach, J. M. Bern, J. Kim, and K. Yamane, "Computational design of robotic devices from high-level motion specifications," *IEEE Transactions on Robotics*, vol. 34, no. 5, pp. 1240–1251, 2018.

[5] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "Voxposer: Composable 3d value maps for robotic manipulation with language models," *arXiv preprint arXiv:2307.05973*, 2023.

[6] Y. Zhu, S. Qiao, Y. Ou, S. Deng, N. Zhang, S. Lyu, Y. Shen, L. Liang, J. Gu, and H. Chen, "Knowagent: Knowledge-augmented planning for llm-based agents," *arXiv preprint arXiv:2403.03101*, 2024.

[7] S. H. Vemprala, R. Bonatti, A. Bucker, and A. Kapoor, "Chatgpt for robotics: Design principles and model abilities," *Ieee Access*, 2024.

[8] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[9] G. Chen, S. Dong, Y. Shu, G. Zhang, J. Sesay, B. F. Karlsson, J. Fu, and Y. Shi, "Autoagents: A framework for automatic agent generation," *arXiv preprint arXiv:2309.17288*, 2023.

[10] Y. Liang, C. Wu, T. Song, W. Wu, Y. Xia, Y. Liu, Y. Ou, S. Lu, L. Ji, S. Mao, *et al.*, "Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis," *Intelligent Computing*, vol. 3, p. 0063, 2024.

[11] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24824–24837, 2022.

[12] C. Li, J. Liang, A. Zeng, X. Chen, K. Hausman, D. Sadigh, S. Levine, L. Fei-Fei, F. Xia, and B. Ichter, "Chain of code: Reasoning with a language model-augmented code emulator," *arXiv preprint arXiv:2312.04474*, 2023.

[13] Q. Dong, L. Li, D. Dai, C. Zheng, J. Ma, R. Li, H. Xia, J. Xu, Z. Wu, T. Liu, *et al.*, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.

[14] Z. Nie, R. Zhang, Z. Wang, and X. Liu, "Code-style in-context learning for knowledge-based question answering," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 18833–18841, 2024.

[15] S. Farritor and S. Dubowsky, "On modular design of field robotic systems," *Autonomous Robots*, vol. 10, pp. 57–65, 2001.

[16] S. Farritor, S. Dubowsky, N. Rutman, and J. Cole, "A systems-level modular design approach to field robotics," in *Proceedings of IEEE International Conference on Robotics and Automation*, vol. 4, pp. 2890–2895, IEEE, 1996.

[17] C. H. Belke and J. Paik, "Mori: a modular origami robot," *IEEE/ASME Transactions on Mechatronics*, vol. 22, no. 5, pp. 2153–2164, 2017.

[18] A. Castano, A. Behar, and P. M. Will, "The conro modules for reconfigurable robots," *IEEE/ASME transactions on mechatronics*, vol. 7, no. 4, pp. 403–409, 2002.

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

[20] R. Desai, Y. Yuan, and S. Coros, "Computational abstractions for interactive design of robotic devices," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1196–1203, 2017.

[21] V. Megaro, B. Thomaszewski, M. Nitti, O. Hilliges, M. Gross, and S. Coros, "Interactive design of 3d-printable robotic creatures," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 6, pp. 1–9, 2015.

[22] J. H. Lee, H. Oh, J. Yoon, S.-J. Lee, T. Jin, J. Hwangbo, and S.-H. Bae, "Robotsketch: an interactive showcase of superfast design of legged robots," in *ACM SIGGRAPH 2024 Emerging Technologies*, pp. 1–2, 2024.

[23] L. Makatura, M. Foshey, B. Wang, F. HähnLein, P. Ma, B. Deng, M. Tjandrasuwita, A. Spielberg, C. E. Owens, P. Y. Chen, *et al.*, "How can large language models help humans in design and manufacturing?," *arXiv preprint arXiv:2307.14377*, 2023.

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

[25] N. Bezzo, A. Mehta, C. D. Onal, and M. T. Tolley, "Robot makers: The future of digital rapid design and fabrication of robots," *IEEE Robotics & Automation Magazine*, vol. 22, no. 4, pp. 27–36, 2015.

[26] R. Ryan, "Adams—multibody system analysis software," *Multibody systems handbook*, pp. 361–402, 1990.

[27] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

[28] S. G. Hart, "Nasa-task load index (nasa-tlx); 20 years later," in *Proceedings of the human factors and ergonomics society annual meeting*, vol. 50, pp. 904–908, Sage publications Sage CA: Los Angeles, CA, 2006.

[29] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[30] S. Kambhampati, "Can large language models reason and plan?," *Annals of the New York Academy of Sciences*, vol. 1534, no. 1, pp. 15–18, 2024.