# 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. Current methods to automate robot design process still require significant human effort and technical expertise. And most methods focus only on a specific design task. This paper present a novel approach for modular robot design using Large Language Models (LLMs) as intelligent agents. 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 textual descriptions, users can generate robot designs that cover a wide range of design requirements. Based on this method, we propose a new text-based interactive modular robot design tool, which can lower the design barrier for non-expert users and improve design efficiency. Further information can be found at: https://fodechain.github.io/ChatBuilder/**

## I. INTRODUCTION

Abstracting the task of robot design into the selection and assembly of modular units[9], [25] can reduce the design space, simplify the design process, and provide a strong foundation for automating robot design. However, most existing work is limited to single design tasks. For example, previous work[18], [31] has optimized mobile robots for different terrains or searched for suitable robotic arms based on end-effector trajectories[8], but it struggles to handle tasks with multiple design requirements simultaneously. Additionally, although existing work[16] has achieved a certain level of automation and simplification in the design process, it still requires significant manual effort and expertise, which not only increases design time and cost but also limits the possibility of non-expert users participating in robot design.

Imagine effortlessly saying, "Give me a robot to help me pick up trash," and receiving a fully manufacturable robot design within minutes. The universality and intuitiveness of expressing design requirements through text could make robots more integrated into our daily lives. Recently, Large Language Models (LLMs) have demonstrated remarkable effectiveness in understanding natural language and handling complex tasks[19], [32], [29]. Their success in various applications suggests that LLMs can be a powerful tool for simplifying the design process of modular robots. Based on this idea, we propose a novel approach to designing modular robots by leveraging LLMs as intelligent agents. With only few-shot in-context learning, our intelligent agent can understand and analyze input text, select appropriate components, and ultimately assemble modular robots that meet the requirements. Building on this method, we develop a new text-based interactive design tool and compare it with the traditional graphical user interface (GUI)-based interaction approach.

Unlike previous work[18], [31], [8] that focus on improving design performance for specific tasks, our work aims to find a more universal and simpler design method to lower the barrier for non-expert users. We validate the versatility of ChatBuilder by inputting various design requirements and conduct case studies for specific design needs. We verify the effectiveness of ChatBuilder's generated results in simulation environments and also prototype some designs to demonstrate their manufacturability and physical feasibility.

## II. RELATED WORK

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

*a) LLMs:* Leveraging GPT-4's powerful multi-modal processing capabilities [1], numerous works have focused on using LLMs as agents to perform complex planning or generation tasks[6], [19], [24]. Chain-of-Thought [30] and Chain-of-Code [22] 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[11]. Previous work[26] 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 case studies 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. In early research [12], [13], a predefined module library, including joints, links, power units, etc., is used to assemble robots. Similarly, [3], [5] also introduce a module library. These moudules 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

*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

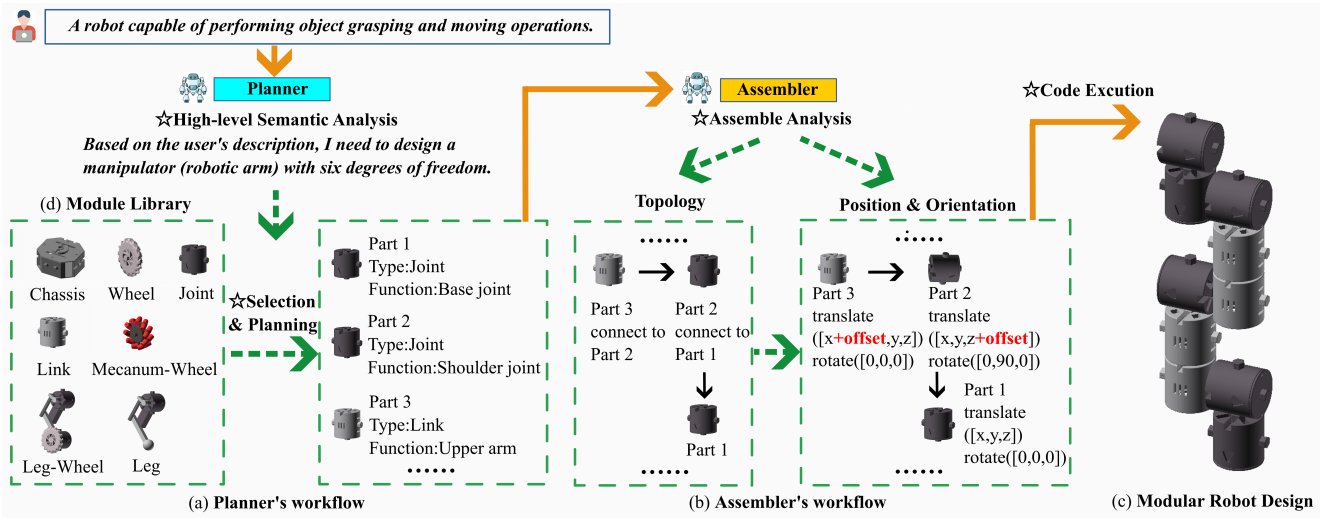



Fig. 1: Overview of ChatBuilder. ChatBuilder consists of two agents: Planner and Assembler. The user only needs to input requirements in text form. The Planner is responsible for performing high-level semantic analysis of the input text, identifying the necessary components, their functions in the design. 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. By executing the code, we can obtain modular robot design that meet the design requirements.

from a library to meet the specific needs of a given task. Evolutionary algorithms have been widely used in modular design [2]. There are also some search based optimization methods or machine learning-based methods, such as terrain based optimization of robot structure [31], [18], robot design based on end trajectory[8], [16]. However, these methods often require multiple iterations of optimization and high computational power. In addition, while these jobs show excellent performance on design tasks, they tend to focus on a specific task.

Some research has explored more interactive ways to design robots [9], [25], where users can quickly design robots by dragging graphical symbols. In[21], 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. Similar to the goal of[9], we also wanted to find a common way to customize robots quickly and easily. But what's different is that we're trying to take advantage of the generality of text to come up with a new text-based interaction that allows users to design modular robots just by inputting text.

## III. METHOD

Recent work explores the possibility of using LLM to assist in robot design[28], which can plan robot components and offer useful suggestions. Our goal is to have the LLM generate robot design that meet the input requirements and can be directly used for manufacturing. This requires the LLM to make more comprehensive and in-depth analyses and produce more standardized outputs. Although research[20] shows that LLMs may not be particularly strong in logical reasoning, many works have still improved their ability to handle complex tasks in specific domains through in-context learning[14], [23], and intelligent agents built on this foundation also demonstrate impressive abilities in handling complex tasks.

This leads us to consider: **Can we enhance the LLM's robot design capabilities through excellent robot design examples in context-based learning?**

To realize this idea, we first need some examples that are easy for the LLM to understand and learn. Inspired by prior work[9], we abstract robot design as the selection and assembly of modular components, and the modular components need to be easily described in text with strong interpretability. To this end, we specifically design a module library and inform the LLM about the functions, dimensions, and assembly rules of the components within the library, which essentially informs the LLM about the design space. Afterward, we build modular robot examples using this module library and link the examples to corresponding design requirements. LLM excel at structured output[26], especially code-style output. Therefore, we represent the examples in code format.

However, our test results show that LLM can only complete design tasks similar to those in the examples. It performs poorly on advanced tasks not encountered in the examples, as detailed in *Ablation* part of the paper. In our analysis, compared to scenarios without example and module library information, LLM, although adhering to the output format, may be limited in its generalization ability. In the examples, there is a direct correspondence between design requirements and output code, which in fact skips the intermediate thinking and analysis, artificially accelerating the LLM's thinking process.

Previous studies have proven that slow thinking through a chain of thought[30] is more suitable for handling complex tasks. Therefore, in order to standardize outputs while enhancing LLM's understanding and analysis of design requirements, we design two agents: Planner and Assembler. Fig 1 shows an overview of our method. 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 section.

### A. Module Library

Before all the work, we first need to establish a module library. OpenSCAD[15] 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 unit 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(d). In addition to these basic components, we also add high-level components to the module library, including: Leg, Leg-wheel, and Mecanum-wheel. The purpose of this is to expand the design space and avoid limiting the Planner's design capability due to constraints in the module library, which could result in design failure.

**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:*<br>*//function: a base for the robotic arm*<br>*//type: chassis*<br>*part_2:*<br>*//function: Base Rotation Joint.*<br>*Allows the robotic arm to rotate around the vertical axis of its base.*<br>*//type: joint*<br>*……* |

Fig. 2: Planner's example.

### B. Planner

The Planner plays an important 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, we need to ensure that the Planner's planning is carried out within a certain set of standards and within the design space we define. Therefore, we inform the Planner about the components in the module library, including their names and dimensions. Since different components serve different functions in different tasks and robots, we inform the Planner of the components' functionalities through two examples. The two examples are a manipulator and a mobile robot. Each example includes a Demo Input and a Demo Output, Fig 2 shows part of a Planner's example. The Demo Input refers to the high-level design requirements. The Demo Output consists of two parts: the first is a process of analyzing the design requirements, including classification of the required robot, extraction and analysis of key information, and even some quantitative calculations and reasoning processes, which we will discuss in more detail in *Case Study* part; the second part is a structured list of components, including the ID of each required component, the function of each component in this specific design task, and the type of each component.

**Assembler's example 1 (manipulator)**

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

Fig. 3: Assembler's example.

### C. Assembler

The input to the Assembler is the output from the Planner. It is responsible for converting the Planner's planning results into assembly code. Specifically, the Assembler needs to analyze the topological relationships between components and the pose information of each component, as shown in Fig 1(b). Compared to the Planner, the Assembler needs to assemble the modular robot according to the given assembly rules. Specifically, it needs to provide the pose of each component, which involves the specific coordinates and orientation of each component. For the coordinates, we use a recursive approach to derive these coordinates, where the position of each component is computed based on the position of the preceding component plus a specified offset. The offset can be along multiple coordinate axes or a single coordinate axis. Each component has multiple different connection interfaces, and the way components are connected depends on the specific task and the function of the component in that task, which needs to be extracted from the Planner's output. Similar to the Planner, we also provide two assembly examples, a manipulator and a mobile

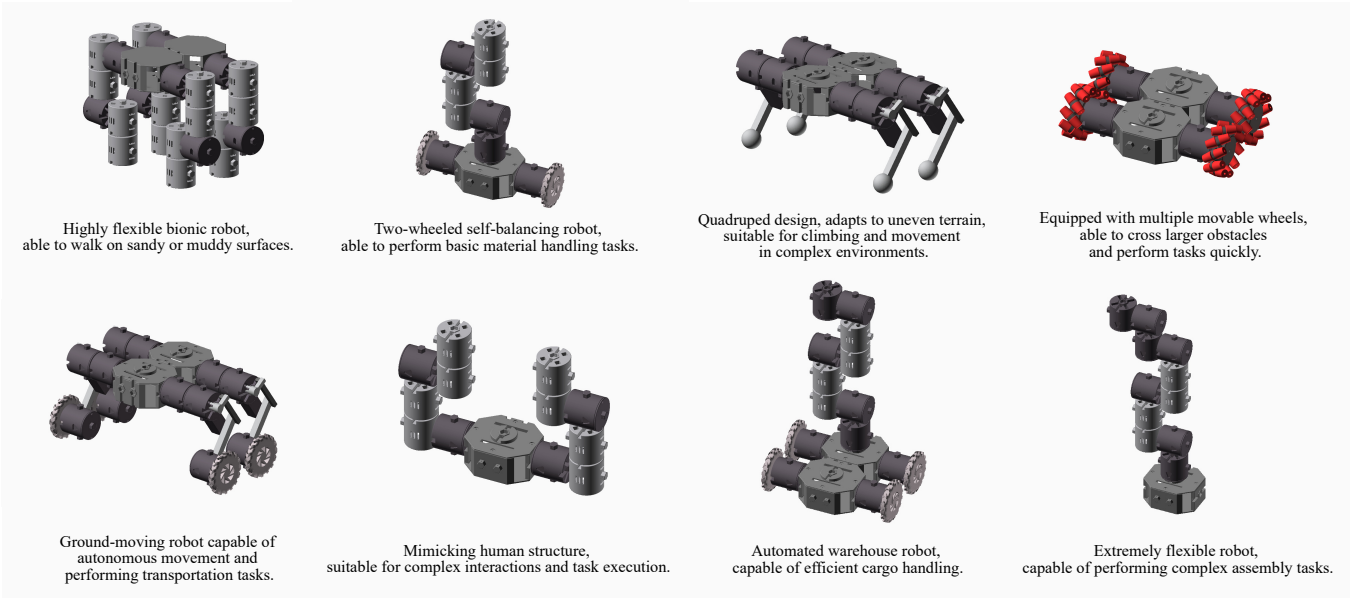



Fig. 4: Qualitative generated results of our approach. Based on text input, our approach can cover a wide range of design requirements and generate diversity robot designs that meet the requirements.

robot, for the Assembler to learn from, as shown in Fig 3. Each example includes a Demo Input and a Demo Output. The Demo Input is the Demo Output from the Planner, and the Demo Output is the assembly code, which includes the component types and pose analysis expressed in comments.

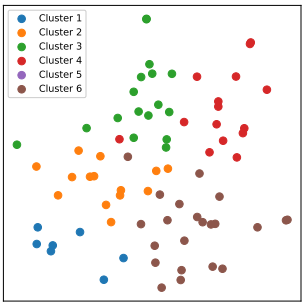



Fig. 5: Visual distribution of test texts. Each point in the figure represents a text, and we cluster them using the K-means algorithm. Different colors represent different categories. Our 78 test texts are evenly distributed and cover at least six different design requirement categories.

## IV. EXPERIMENTS

Our experiments aim to address the following questions:

Q1: Can ChatBuilder cover a wide range of design requirements?

Q2: How does ChatBuilder perform in completing specific tasks?

Q3: Can the text-based interaction approach improve efficiency and lower the barrier to robot design work?

Previous robot design efforts have mostly focused on specific individual tasks[8], [31]. The evaluation methods and metrics vary significantly across different tasks, making it challenging to establish a fair baseline for comparison. In contrast, our work aims to cover as many design requirements as possible, rather than focusing solely on a single specialized task. This broad scope, however, results in a lack of directly comparable baselines for evaluation. Furthermore, compared to design performance on a single task, we place greater emphasis on whether ChatBuilder can fulfill as many design requirements as possible while remaining consistent with the textual semantics.

Therefore, based on the broad robot design principles mentioned in[4], we evaluate the robot designs generated by ChatBuilder according to the following criteria:

Assembly Validity: whether the components are free from interference or overlap, and whether they are connected through the correct interfaces. This is verified using Adams[27], a multi-body dynamics simulation software.

Requirement Compliance: whether the designed modular robot adheres to the structural or functional specifications described in the text. This is determined through a combination of simulation and expert judgment.

In collaboration with websites and prior work[4], we collect and summarize several robot design requirements described in text. We exclude those requirements that fall outside our design space, as well as duplicate or similar design requirements, resulting in 78 text entries, which we use as test texts. Some of the test texts can be found in Fig 4. It is important to note that, rather than focusing on the quantity of the test texts, we are more concerned with whether they cover a broader range of design requirements, in order to validate the design scope of ChatBuilder. We use BERT[10] to vectorize these texts and compress them into a two-dimensional space for visualization, as shown in Fig 5. As can be seen, they are evenly distributed and cover multiple categories.

### A. Results

Fig 4 presents the qualitative generated results of our approach.

For each text, we repeat the process 20 times and record the number of times ChatBuilder generate results that are assembly-effective or meet the requirements. And then divide these counts by 20 to calculate the success rate. Furthermore, we also calculate the cosine similarity between each text and the Demo Input used for in-context learning. As described in *Method* part of this paper, we have two Demo Inputs, and the cosine similarity here refers to the similarity between the text and the closest Demo Input of Planner. As shown in Fig 6, when the similarity between the test text and the Demo Input is high, ChatBuilder shows a higher success rate in both assembly-effective and meeting the requirements. When the similarity is low, ChatBuilder may still complete the design with a high success rate. 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. We also find that Assembly Validity and Requirement Compliance are consistent in the vast majority of cases. However, when the similarity between the input text and the Demo Input is low, more results are generated that satisfy Assembly Validity but do not meet Requirement Compliance.

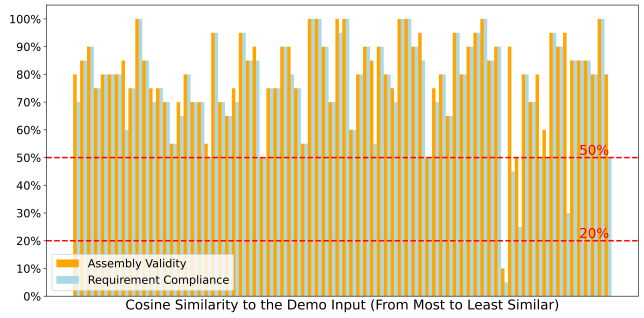



Fig. 6: Quantitative results of ChatBuilder on all 78 test texts. Orange bars represent the success rate of generated results being assembly-effective across all 20 tests, while blue bars represent the success rate of generated results meeting the requirements across all 20 tests. It can be seen that lower success rates are more likely to occur when the similarity between the test text and the Demo Input is low.

Among all the 20 repeated tests, the average number of assembly-effective results is 16, and the average number of results that meet the requirements is 15. We use whether the results meet the requirements (Requirement Compliance) as the final evaluation criterion. In other words, ChatBuilder can generate modular robot designs that meet the textual requirements with an average success rate of $75\%$.

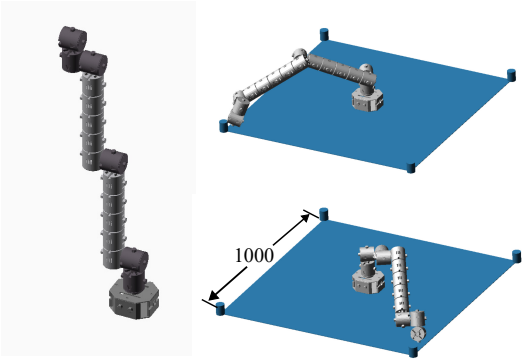
### B. Case Study

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.***

ChatBuilder generate a six-degree-of-freedom robotic arm. Not only that, but its position is moved from the origin (0, 0, 0) to (500, 500, 0), the coordinates here represent (x, y, z). And the number of links composing the upper arm and lower arm is increased to five. We simulate the generated results, and the robotic arm is indeed able to reach any corner of the square room, as shown in Fig 7.




(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.***"

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.***

Given the dimensions of the components, ChatBuilder is able to perform simple computational reasoning based on the textual requirements and ultimately determine the number of components to select—all without any reference to a demo. Fig 8 also shows the generated results for other input texts containing quantitative constraints.

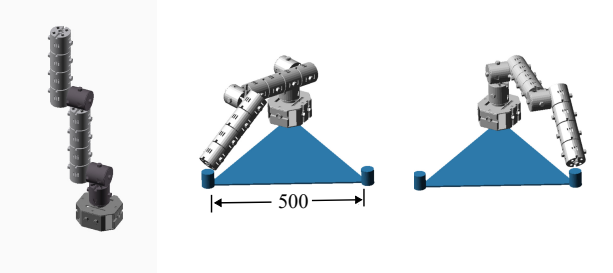
Additionally, we find that ChatBuilder can generate diverse results for the same text requirement. Fig 9 shows one such example. ChatBuilder can generate a diverse results while ensuring that the output meets the textual requirements. This demonstrates that Chatbuilder has both robustness and diversity.

### C. User Study

Previous work mainly design the robot through GUI[9], [25], and to verify whether the text-based interaction approach in ChatBuilder can enhance design efficiency and lower the design barrier, we conduct the following comparative experiment:

We invite 10 experts and 10 non-experts, with the experts possessing extensive modeling experience and specialized knowledge in robotics design, while the non-experts have no prior experience in modeling or knowledge related to robotics design. They are asked to perform the same design

***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.***

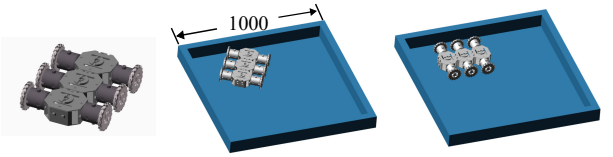



Fig. 8: Generated results and simulation for other input texts containing quantitative constraints.

***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.

task using both interaction methods. For each participant, we randomly select a text from the 78 test texts, and the users are allowed to use any component from the module library to design a modular robot that meets the requirements. To ensure fairness, we provide both diagram and code versions of the Demo used for in-context learning to both experts and non-experts, helping them gain a fundamental understanding of the module library and design space.

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. In practical applications, users can repeatedly invoke ChatBuilder within a short time frame to meet their design requirements.

We also design a questionnaire based on NASA-TLX[17] and collect feedback from participants regarding the workload of two different interaction approaches. As shown in Fig 10, both experts and non-experts find that using ChatBuilder for design tasks results in a lower workload, with this being particularly evident among non-experts.

This demonstrate that ChatBuilder can improve design efficiency and lower the design barrier for non-experts.

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

TABLE I: Participants' time usage. The first column indicates the average time taken to successfully complete a design using the GUI, the second column indicates the average time taken to successfully complete a design using ChatBuilder, and the third column shows the average number of times ChatBuilder was used during design process. When using ChatBuilder for robot design, both experts and non-experts require less time.

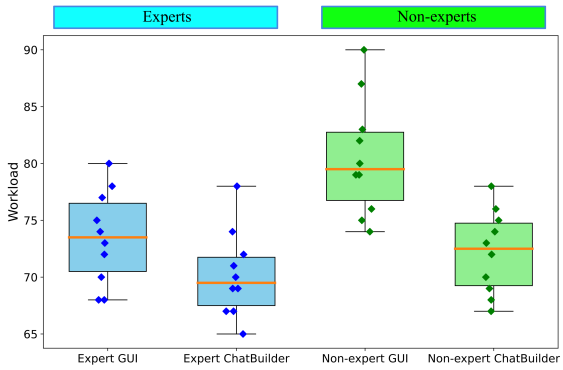



Fig. 10: Participant feedback on the workload of two different interaction methods. Both experts and non-experts report a lower workload when using ChatBuilder for design tasks, with a more significant difference observed among non-experts.

## D. Ablation

In the ablation study, we still use test texts for the trials, repeat each text 20 times, and calculate the average number of results that met the requirements (Requirement Compliance) to determine the success rate. We also count the average number of times different generated results are produced to assess Diversity.

We 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.

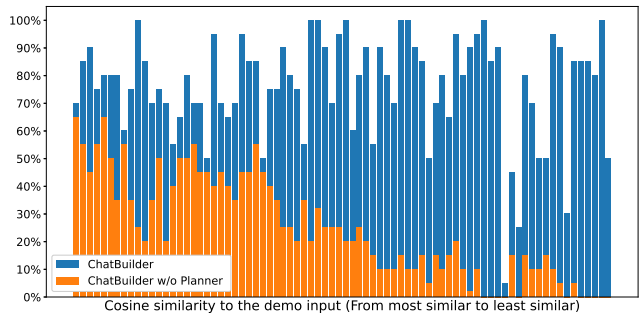



Fig. 11: The absence of a Planner significantly reduces ChatBuilder's success rate in Requirement Compliance, especially when handling more challenging tasks (where the similarity between the input text and the Demo Input is low), to the point where the task can not even be completed.

We try using a single agent to directly learn the final assembly code through context, eliminating the Planner. As shown in Fig 11, 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. This is especially evident in the design of advanced tasks.

We also test the impact of different numbers of Demos on the generated results. Fig 12 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[7]: more Demos exceed the input length limit of the LLMs, causing the Demos to be truncated and reducing the success rate. For Diversity, when the number of Demos exceeds 2, the Diversity of 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.

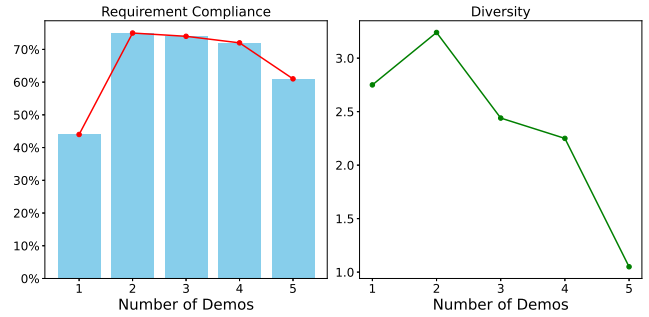



Fig. 12: As the number of Demos increases, the success rate of ChatBuilder in Requirement Compliance initially rises and then levels off. When the number of Demos reaches 5, the success rate actually declines, which is due to the truncation problem. In terms of Diversity, when the number of Demos is 2, approximately 3 different generated results can be found in 20 repetitions. However, as the number of Demos continues to increase, the diversity of the generated results decreases, as ChatBuilder tends to directly imitate the Demos rather than create its own variations.

### *E. Real World Experiment*

We build some prototypes based on the design diagrams generated by ChatBuilder. We select a 42 stepper motor as the drive for the real-world robot. 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 13(a). For the mobile robot generated by ChatBuilder, it smoothly navigate around obstacles and pass through narrow passages, as shown in Fig 13(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. CONCLUSIONS

In this paper, we propose a new approach for modular robot design using LLMs. It is demonstrated through experiments that ChatBuilder, with just few-shot in-context learning, can generate modular robots that meet design requirements based on input text. Thanks to the universality of text-based expression and the reasoning and planning capabilities of LLMs, ChatBuilder can cover a wide range of design requirements, and the generated results exhibit robustness and diversity. At the same time, ChatBuilder can also accomplish some advanced tasks with quantitative constraints. We develop a new text-based interactive tool based on this method, which, compared to traditional GUIs, can improve design efficiency and lower the design barrier for non-experts.

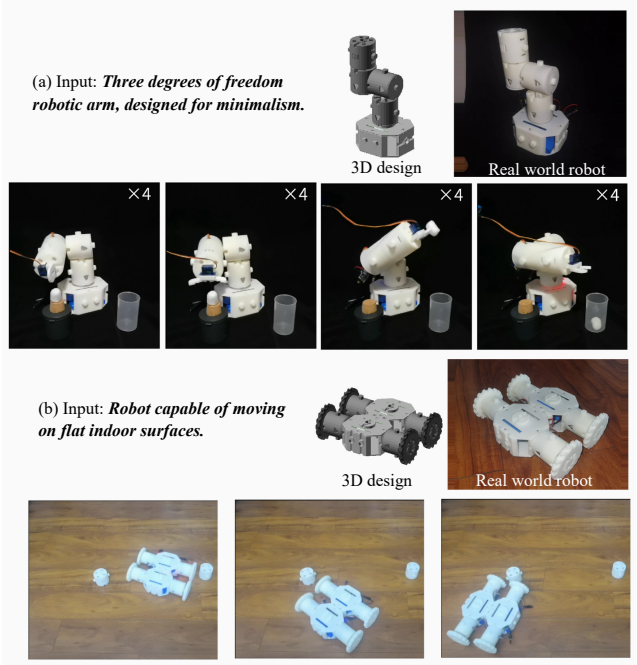



Fig. 13: Real world experiments. We build several prototypes and use them to complete the tasks.

However, it still has some limitations: First, although ChatBuilder can handle some advanced tasks with quantitative constraints, its success rate is lower than that for lower-level tasks. More exploration is needed to improve its design performance for advanced tasks. Second, although text is already a relatively universal form of expressing design requirements, there are still some design requirements that can not be accurately expressed through text alone, such as the robot's motion and movement trajectory, which makes it difficult for ChatBuilder to handle this type of task. Finding a more universal and accurate representation of robot design requirements will be key to solving this problem.

## REFERENCES

[1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[2] Reem J Alattas, Sarosh Patel, and Tarek M Sobh. Evolutionary modular robotics: Survey and analysis. *Journal of Intelligent & Robotic Systems*, 95:815–828, 2019.

[3] Christoph H Belke and Jamie Paik. Mori: a modular origami robot. *IEEE/ASME Transactions on Mechatronics*, 22(5):2153–2164, 2017.

[4] Nicola Bezzo, Ankur Mehta, Cagdas Denizel Onal, and Michael Thomas Tolley. Robot makers: The future of digital rapid design and fabrication of robots. *IEEE Robotics & Automation Magazine*, 22(4):27–36, 2015.

[5] Andres Castano, Alberto Behar, and Peter M Will. The conro modules for reconfigurable robots. *IEEE/ASME transactions on mechatronics*, 7(4):403–409, 2002.

[6] Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F Karlsson, Jie Fu, and Yemin Shi. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*, 2023.

[7] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[8] Ruta Desai, Margarita Safonova, Katharina Muelling, and Stelian Coros. Automatic design of task-specific robotic arms. *arXiv preprint arXiv:1806.07419*, 2018.

[9] Ruta Desai, Ye Yuan, and Stelian Coros. Computational abstractions for interactive design of robotic devices. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1196–1203, 2017.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina 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)*, pages 4171–4186, 2019.

[11] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Tianyu Liu, et al. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*, 2022.

[12] Shane Farritor and Steven Dubowsky. On modular design of field robotic systems. *Autonomous Robots*, 10:57–65, 2001.

[13] Shane Farritor, Steven Dubowsky, Nathan Rutman, and Jeffrey Cole. A systems-level modular design approach to field robotics. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 4, pages 2890–2895. IEEE, 1996.

[14] Shuzheng Gao, Xin-Cheng Wen, Cuiyun Gao, Wenxuan Wang, Hongyu Zhang, and Michael R Lyu. What makes good in-context demonstrations for code intelligence tasks with llms? In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 761–773. IEEE, 2023.

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

[16] Sehoon Ha, Stelian Coros, Alexander Alspach, James M Bern, Joohyung Kim, and Katsu Yamane. Computational design of robotic devices from high-level motion specifications. *IEEE Transactions on Robotics*, 34(5):1240–1251, 2018.

[17] Sandra G Hart. Nasa-task load index (nasa-tlx); 20 years later. In *Proceedings of the human factors and ergonomics society annual meeting*, volume 50, pages 904–908. Sage publications Sage CA: Los Angeles, CA, 2006.

[18] Jiaheng Hu, Julian Whitman, Matthew Travers, and Howie Choset. Modular robot design optimization with generative adversarial networks. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4282–4288. IEEE, 2022.

[19] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023.

[20] Subbarao Kambhampati. Can large language models reason and plan? *Annals of the New York Academy of Sciences*, 1534(1):15–18, 2024.

[21] Joon Hyub Lee, Hyunsik Oh, Junwoo Yoon, Seung-Jun Lee, Taegyu Jin, Jemin Hwangbo, and Seok-Hyung Bae. Robotsketch: an interactive showcase of superfast design of legged robots. In *ACM SIGGRAPH 2024 Emerging Technologies*, pages 1–2. 2024.

[22] Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. Chain of code: Reasoning with a language model-augmented code emulator. *arXiv preprint arXiv:2312.04474*, 2023.

[23] Jia Li, Ge Li, Chongyang Tao, Huangzhao Zhang, Fang Liu, and Zhi Jin. Large language model-aware in-context learning for code generation. *arXiv preprint arXiv:2310.09748*, 2023.

[24] Yaobo Liang, Chenfei Wu, Ting Song, Wenshan Wu, Yan Xia, Yu Liu, Yang Ou, Shuai Lu, Lei Ji, Shaoguang Mao, et al. Taskmatrix. ai: Completing tasks by connecting foundation models with millions of apis. *Intelligent Computing*, 3:0063, 2024.

[25] Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3d-printable robotic creatures. *ACM Transactions on Graphics (TOG)*, 34(6):1–9, 2015.

[26] Zhijie Nie, Richong Zhang, Zhongyuan Wang, and Xudong Liu. Code-style in-context learning for knowledge-based question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18833–18841, 2024.

[27] RR Ryan. Adams—multibody system analysis software. *Multibody systems handbook*, pages 361–402, 1990.

[28] Francesco Stella, Cosimo Della Santina, and Josie Hughes. How can llms transform the robotic design process? *Nature machine intelligence*, 5(6):561–564, 2023.

[29] Sai H Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. *Ieee Access*, 2024.

[30] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

[31] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. Robogrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics (TOG)*, 39(6):1–16, 2020.

[32] Yuqi Zhu, Shuofei Qiao, Yixin Ou, Shumin Deng, Ningyu Zhang, Shiwei Lyu, Yue Shen, Lei Liang, Jinjie Gu, and Huajun Chen. Knowagent: Knowledge-augmented planning for llm-based agents. *arXiv preprint arXiv:2403.03101*, 2024.