Alchemist: LLM-Aided End-User Development of Robot Applications

Ulas Berk Karli
ulasberk.karli@yale.edu
Yale University
New Haven, CT, USA

Juo-Tung Chen
jchen396@jhu.edu
Johns Hopkins University
Baltimore, MD, USA

Victor Nikhil Antony
vantony1@jhu.edu
Johns Hopkins University
Baltimore, MD, USA

Chien-Ming Huang
cmhuang@cs.jhu.edu
Johns Hopkins University
Baltimore, MD, USA

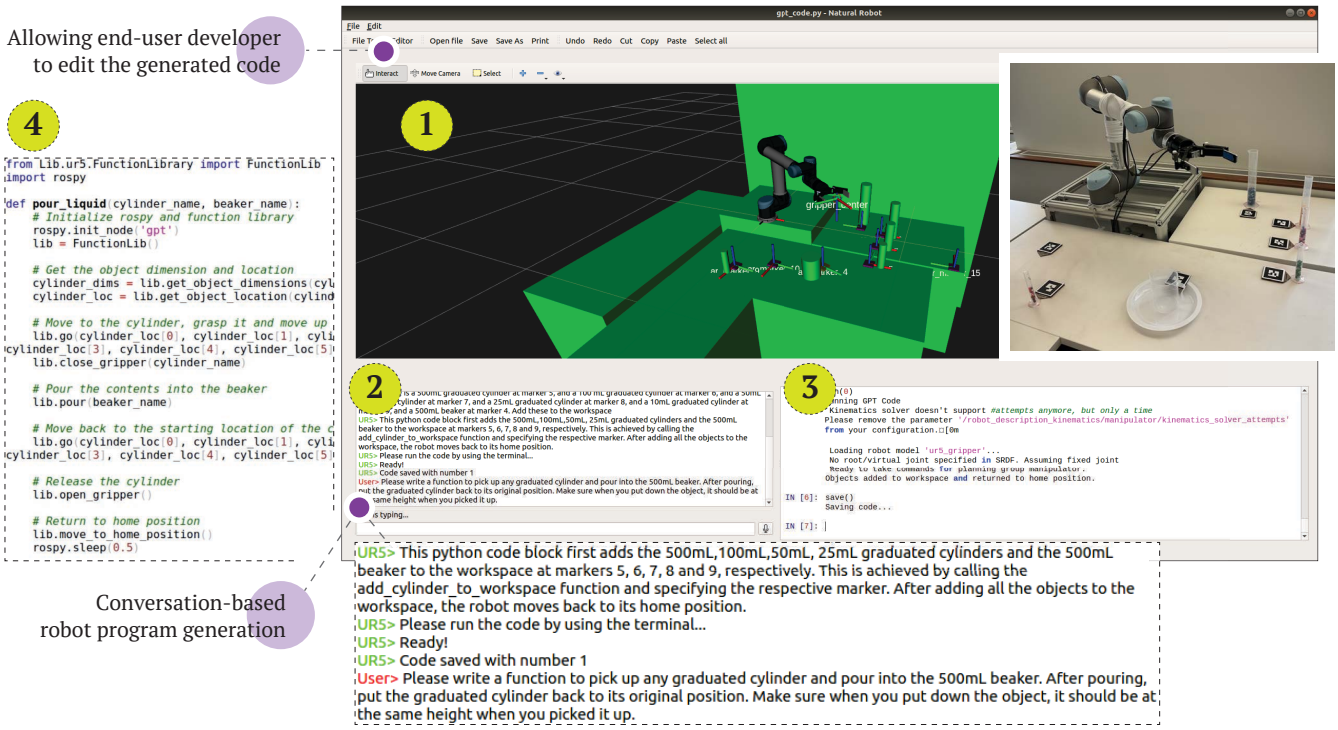


Figure 1: *Alchemist* is an end-to-end end-user robot programming system that leverages large language models (LLM) to enable natural language based robot program authoring. 1) The 3D RViz Visualization Panel visualizes the robot and its environment. 2) Users interact with a LLM to create robot programs in the chatbox either through text or voice inputs. 3) Saved programs can be executed using the terminal panel. 4) Users can also directly edit the generated programs for finer control.

ABSTRACT

Large Language Models (LLMs) have the potential to catalyze a paradigm shift in end-user robot programming—moving from the conventional process of user specifying programming logic to an iterative, collaborative process in which the user specifies desired program outcomes while LLM produces detailed specifications. We introduce a novel integrated development system, *Alchemist*, that leverages LLMs to empower end-users in creating, testing, and running robot programs using natural language inputs, aiming to reduce the required knowledge for developing robot applications. We present a detailed examination of our system design and provide an exploratory study involving true end-users to assess capabilities, usability, and limitations of our system. Through the design, development, and evaluation of our system, we derive a set of lessons learned from the use of LLMs in robot programming. We discuss how LLMs may be the next frontier for democratizing end-user development of robot applications.

HRI '24, March 11–14, 2024, Boulder, CO, USA

ACM ISBN 979-8-4007-0322-5/24/03.
https://doi.org/10.1145/3610977.3634969

CCS CONCEPTS

• **Human-centered computing**; • **Computer systems organization** → **Robotics**;

KEYWORDS

robot programming, end-user development, code generation

ACM Reference Format:
Ulas Berk Karli, Juo-Tung Chen, Victor Nikhil Antony, and Chien-Ming Huang. 2024. Alchemist: LLM-Aided End-User Development of Robot Applications. In *Proceedings of the 2024 ACM/IEEE International Conference on Human-Robot Interaction (HRI '24), March 11–14, 2024, Boulder, CO, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3610977.3634969

1 INTRODUCTION

Robots have the potential to significantly boost productivity across diverse fields from factory floors to research laboratories. Yet, deploying robots can be complex and costly, often requiring teams of experts for system initialization and task-specific programming. Consider a scenario where a team of researchers is striving to enhance the efficiency of photocatalysts for hydrogen production from water, a pursuit often described as the "holy grail" of chemistry [48]. This endeavor may require extensive experimental exploration of a diverse set of candidates and configurations; for instance, evaluating a search space with ten variables could present approximately 98 million potential candidates. Even with optimization strategies applied to narrow down this search space, as many as 688 experiments might still be required to identify superior photocatalyst mixtures. Robotic chemist [10] performed these experiments and exemplified how robots can automate the repetitive, low-level testing procedures, liberating scientists to focus on high-level tasks.

Nevertheless, robots deployed in such specialized roles are often laboriously developed for specific tasks and environments, posing a barrier to the widespread integration of robots in complex and dynamic workflows such as life science laboratories. Unlike computers, which can be readily acquired and customized with various software applications, the accessibility and customizability of robotic platforms and programs by end-users, such as scientists, remain elusive. To democratize the use of robots, several end-user programming systems and frameworks have been proposed [2]. Approaches for lowering barriers to programming robots include utilizing behaviour trees [38], illustration-based systems [40, 41], flow-based systems [5], programming by demonstration [3, 24], block-based systems [43] and a mix of methods that utilize high-level coding abstractions to program robots [23]. However, these abstractions still require an understanding and explicit specification of the programming logic which presents a significant barrier.

Large Language Models (LLMs), with their ability to generate code from natural language inputs [17], could help catalyze a paradigm shift in end-user robot programming from the conventional process of user specifying programming logic to an iterative, collaborative process in which the user specifies desired program outcomes while LLM produces detailed specifications. This paradigm shift conceptually regards end-user programming systems as collaborators rather than tools. By enabling a more intuitive and collaborative robot programming experience for end-users, LLMs may help lower barriers to the deployment and effective use of robots in diverse settings.

To explore the potential of using LLMs for end-user robot programming, we developed *Alchemist*, an end-to-end system that aims to streamline the complex process of robot programming by empowering users to create, debug, test, and execute robot programs using natural language dialog through a one-stop interface. Alchemist integrates RViz for robot visualization, a chat-box for interacting with the LLM, and a terminal to run generated code (Fig. 1). It is designed to be robot-platform and LLM agnostic to support various settings and technical advancements. It further supports authoring robot applications that involve automated processes (*e.g.,* robotic chemist) and human-robot interaction scenarios.

This work makes three key contributions: **(1)** An open-source, end-to-end system that utilizes LLMs to enable a collaborative and intuitive robot programming experience for end-users. **(2)** An exploratory study to test and understand system capabilities and usability. **(3)** A set of lessons learned to inform the design and development of future LLM-powered robot programming systems.

2 RELATED WORK

Recent strides in LLMs and their code generation capabilities provide an opportunity for developing a new end-user robot programming paradigm. In this section, we summarize prior work and identify gaps in end-user robot programming systems, LLM-enabled code generation and LLMs for robot programming.

2.1 End-User Robot Programming Systems

Robot programming traditionally demands domain-specific expertise [22, 45], posing a barrier for the general population. Several different modes of programming have been explored to enable end-user robot programming [2], aiming to democratize the utilization of robots. A common mode is visual programming including block based programming [4, 15, 23, 53, 59], flow diagrams [7, 19], rule based programming [33], and behaviour trees [8, 39]. Mixed reality-based end-user programming systems have also been developed and evaluated [21, 27, 35]. Apart from these, a hybrid approach utilizing natural language chat and block based programming has been shown to be possible [20]. These existing approaches, while valuable, often still require end-users to possess some knowledge of programming logic, limiting their accessibility to the broader population. The rise of large language models (LLMs) presents an unique opportunity to address the limitation posed by programming logic complexity; by leveraging the capabilities of LLMs, this paper explores how natural language can serve as an intuitive means for individuals to interact with and program robots.

2.2 LLM-enabled Code Generation

As LLMs code generation capabilities have improved significantly, there has been a shift in utilizing them as programming assistants such as GitHub Copilot [17]. Unlike these coding assistants which assist in the process of writing code, our system, *Alchemist*, generates code autonomously as it is designed for users with minimal coding skills. Additionally, Alchemist is tailored specifically for robotics applications. LLMs such as GPT-3 exhibit substantial code generation capabilities [13, 14]; however, there are notable limitations [44, 46]. Usability studies reveal that while LLM-based tools can provide helpful starting points, understanding, editing, and debugging the generated code can be challenging for programmers [50]. Furthermore, evaluations underscore concerns about functional correctness, suggesting that not all code generated by LLMs is error-free [30, 37]. Therefore, there is a need to identify common mistakes made by LLMs [12] and develop a general approach to

rectify these errors effectively. To address errors in generated code, Alchemist includes quality assurance methods (Section 3.3.3).

2.3 LLMs for Robot Programming

The incorporation of LLMs into robot programming has recently gained considerable attention, opening up possibilities for enhancing human-robot interactions. Generating code from natural language descriptions facilitates the control of robots through human-readable instructions. These capabilities have the potential to streamline robot programming. For instance, ChatGPT has been used to generate code for controlling robots using natural language instructions in a zero-shot fashion [51]; however, one limitation of this work is the absence of an integrated code editor and visualization panel, which makes it challenging for users to directly interact with the system and debug errors when they occur. Other studies have also investigated the application of pre-trained LLMs in robotic task planning [16, 25, 49, 55] and reasoning [1, 18, 28, 56]. While these studies demonstrate the potential of LLMs in robotics, they do not provide a straightforward means for end-users to interact with robots or troubleshoot issues in case of errors or unintended actions. In our system, we address these limitations by providing an end-to-end, one-stop system that incorporates a visualization panel and an embedded code editor to facilitate iterative and collaborative interaction between the user, the LLM, and the robot.

Perhaps the work by Inagaki et al. [26] is closet to ours; it explores the integration of LLMs in robot programming, particularly focusing on its application in automating biological laboratories. Although it showcases the potential of using LLMs to enable individuals with limited robotics experience to program robots, it does not fully address the challenge of preventing undesired actions or errors in the code generated by LLMs. Our system addresses this challenge by employing grounded prompting and a code verification process to enhance the code quality.

3 ALCHEMIST: LLM-POWERED END-USER ROBOT PROGRAMMING SYSTEM

Alchemist is an open-source[1], end-to-end system that empowers users to create, edit, and test programs for robots through natural language-based dialog by using large language models as its code generation backbone. We outline our core design objectives (DO) for Alchemist in Table 1 and present an overview of the system followed by details on system implementation in this section.

3.1 System Overview

Using Alchemist, users initiate the robot programming process by placing AR markers to identify objects of interest in robot's workspace. These markers enable the system to track, update, and visualize the robot's world model. Subsequently, users prompt the LLM to generate code via the Chat Panel. They can then edit and debug the generated programs, either by prompting the LLM further or by utilizing the Editor. Once satisfied, users can preview, execute and test the programs via the Terminal Panel.

Alchemist has two distinct task-level capabilities:

[1]https://tinyurl.com/alchemist-github

Table 1: Alchemist's Design Objectives (DO) and Rationales

DO1: Faciliate Programming with Natural Language We seek to use LLMs to facilitate an intuitive robot programming experience through natural language communication, aiming to reduce the need for programmatic thinking of end-users.
DO2: Enable End-to-end Robot Development Workflow Robot programming is a complex multi-step process. We seek to simplify this process by providing a one-stop shop for development, testing, debugging, and execution of robot programs.
DO3: Support Varied Programming Proficiencies End-users of robotic systems have varying programming preferences. We aim to offer a dynamic framework that adjusts the level of code generation abstraction, allowing users to choose between more flexible, general-purpose code that requires increased oversight and debugging, and more rigid, task-specific code that demands minimal oversight.
DO4: Visualize Robot World and Actions Robot programs have real-world physical outcomes which necessitates previewing of program execution for ensuring safety and supporting debugging. We visualize the real-time robot world model and enable preview of actions to prioritize user understanding, control and safety over programs.
DO5: Ensure System Modularity We seek to accommodate various LLMs and robot platforms with ease. Our system-level design ensures adaptability to evolving needs or technological updates without altering the interaction paradigm.

1) Automation: Alchemist allows users to automate entire processes (see Fig. 2-a). This can be done by either breaking tasks into smaller sub-tasks and then creating and integrating individual code segments for each sub-task or by devising a single comprehensive program for the entire task; this choice depends on user preference and task complexity.

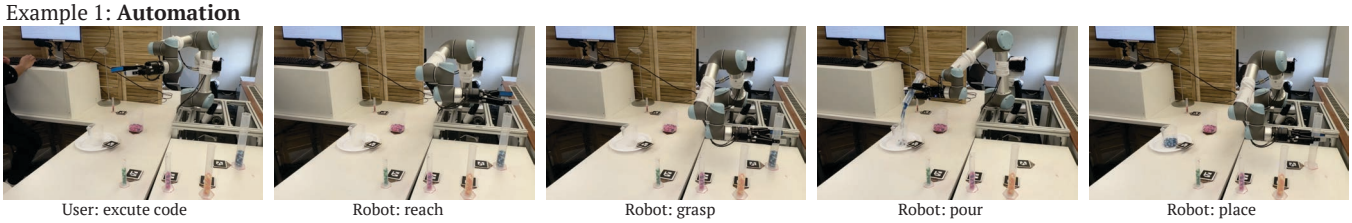
2) Collaboration: Alchemist allows users to program the robot to be collaborative in a way that they can verbally instruct the system to execute specific actions, aiding them in task completion or they can specify how or when they would need the system to do certain actions in response to user actions or changes in environment (see Fig. 2-b).

3.2 Front-End Components

We designed Alchemist's interface (Fig. 1) to be plain and functional to simplify the programming process and lower barriers to robot programming for end-users. It consists of three primary panels: 3D visualization panel, Terminal panel, and Chat panel. Moreover, the system incorporates two supplementary panels (Text Editor and File Tree), accessible to users via toggle buttons (top menu bar).

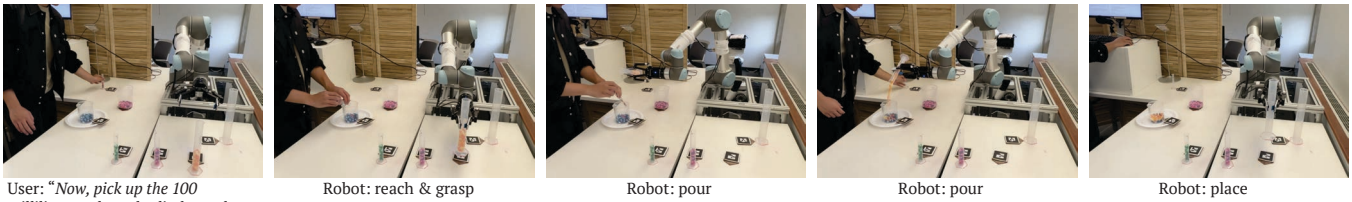
3.2.1 3D Visualization Panel. This panel has an embedded RVIZ interface to visualize the information from the motion planning system regarding the robot's physical environment (see Fig.1-1). Here, users can identify and address discrepancies between the physical

Example 1: **Automation**



User: excute code | Robot: reach | Robot: grasp | Robot: pour | Robot: place

Example 2: **Collaboration** (user verbally requests an assistance)



User: "*Now, pick up the 100 milliliter graduated cylinder and pour into 500 milliliter beaker.*" | Robot: reach & grasp | Robot: pour | Robot: pour | Robot: place

Figure 2: Example development and execution flow of different system capabilities.

and the virtual worlds used in motion planning. Additionally, the panel is designed to be versatile: when paired with a Gazebo ROS node, it can act as a fully simulated environment.

3.2.2 Chat Panel. This panel allows users to interact with the LLM to generate the code required for their tasks (Fig. 1-2). It provides an input box for users to communicate with the LLM and contains the complete chat history. Once the LLM generates code and provides an explanation, only the explanation is presented in the chat history. Additionally, the chat panel allows users to communicate via voice input in addition to manual typing; this is enabled by speech-to-text functionality implemented using OpenAI's Whisper [42].

3.2.3 Terminal Panel. This panel is a Python terminal enriched with built-in functions to let users inspect, run, and save code and reset the system as needed (see Fig.1-3). The built-in functions and their usage descriptions are presented in Appendix A.

3.2.4 Text Editor and File Tree Panels. The text editor constitutes a comprehensive text editing tool enriched with syntax highlighting features tailored for Python code (Fig. 1-4). Meanwhile, the file tree panel exclusively displays files generated by LLM and subsequently saved by the user. Its principal function is to facilitate users in monitoring the quantity of files they have saved, enabling them to rerun these files through the terminal when needed. By default, these panels are hidden, and users can unveil them by selecting the "Editor" or "File Tree" buttons located within the upper horizontal menu bar. This design approach, wherein the panels remain hidden by default, serves a specific purpose: to prevent users from feeling overwhelmed by an excessive array of panels and code.

3.3 Back-End Components

Alchemist's back-end has several components to achieve the desired functionalities: Function Library, LLM Initialization Prompting, and Code Safety Mechanisms. We built upon the design principles laid out in Vemprala et al. [51] for the function library and LLM prompts but with enhanced functionalities tailored for end-user robot programming. Fig. 3 provides a high-level overview of the back-end operations; the implementation details are provided below.

3.3.1 Function Library. The function library is a platform-specific code library designed to be a broad set of tools for the LLM to use. It is a layer of abstraction over underlying ROS functions and services specific to achieve general actions (*e.g.,* moving the robot, operating the gripper). The LLM is provided descriptions of these functions in its initial prompt and is instructed to use only these functions. We assign descriptive names to the functions in this library and their input to further leverage the natural language understanding capabilities of LLMs [51].

Our system is designed to cater to a wide spectrum of users, from those proficient in coding to individuals with little to no coding experience. We achieve this variability in LLM code output based on the chosen abstraction level through our initial prompt, rather than by altering the function library itself. In our current implementation, the function library contains functions with two levels of abstraction: high-level and low-level. This approach makes our implementation of the function library simple, easy to interpret, and interchangeable, resulting in a modular system.

High level abstraction support novice users or those with limited coding experience. These high-level functions are closely interconnected, serving specific, task-oriented purposes, and possess a limited number of input parameters. An example high-level function is *pour(target_name)* which takes the target container name to pour as an input and pours the container that is gripped by the robot into the target container given in the input, as can be seen this is a highly engineered, abstract function. Conversely, low-level functions are more versatile, encompassing a broader range of potential uses and interactions. They boast a greater degree of generality and offer more extensive functionality, while tending to be more error-prone. An example low-level function is *move(x,y,z,roll,pitch,yaw)* which takes the end-effector pose and moves the end-effector to that pose, as can be observed from this example this function is a small wrapper over the underlying robot API and not as abstract;

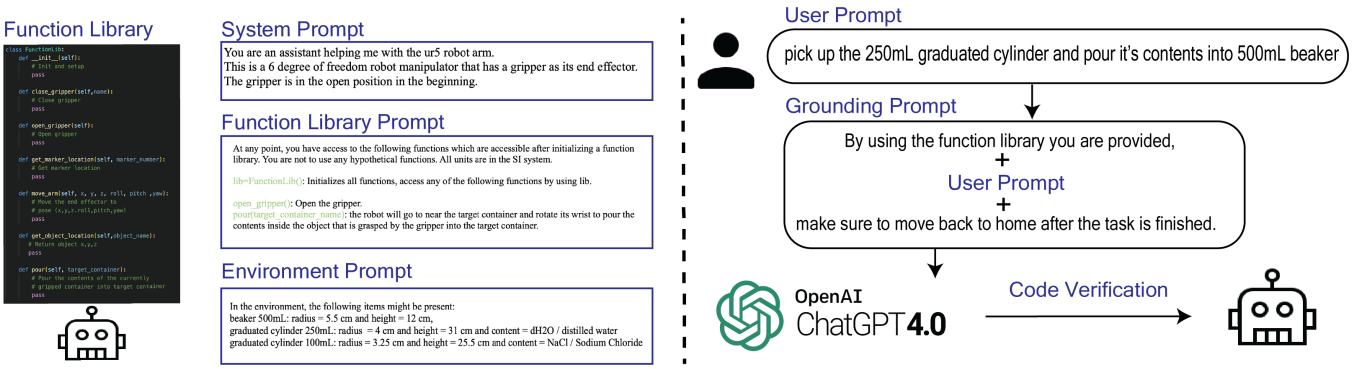


Figure 3: Illustration of back-end components of Alchemist: the function library, initialization prompt and grounded prompts

notice that the move function can enable similar outcome as the *pour(target_name)* function but with more granular control over the specifics of the movement. More examples of different abstraction level libraries can be found in supplementary materials.

3.3.2 Initial Prompting of the LLM. For our initial release and exploratory study, we employed OpenAI's GPT-4 [36]; we chose GPT-4 due to its state-of-the-art reasoning capabilities and simple API usage. The key to effectively leveraging an LLM for programming is initializing the model with well-defined function library, a system role, and an environment prompts [51].

We instruct the LLM to act as a robot programming helper. We also specify the robot in terms of name, number of degrees of freedom, end effector, and type of code that it needs to return with some warnings, rules, and caveats (*e.g.,* "always use floating numbers" and "you are not to use any other hypothetical functions that you think might exist"). The function library prompt contains the names, inputs, functionalities, and outputs of each function that is available to the LLM along with axis and unit conventions; this prompt acts as a descriptive code documentation.

The environment prompt is used to inform the system about the presence of physical objects within the workspace. For our exploratory study, we utilized various-sized beakers and graduated cylinders, all of which are listed in the environment prompt with their respective dimensions. If a sophisticated perception system, such as a vision language model [58], is used, the environment prompt could be omitted.

The final component of the initial prompt is the example user prompt, coupled with the corresponding example code output. This element holds significant importance in enabling the LLM to generate high-quality code. In our exploratory study, the provided example is thoughtfully crafted to incorporate a wide array of functions from the function library. The example given in our system is designed in a way that utilizes most of the functions in the library to reinforce the correct usage of each function by the LLM.

Lastly, to overcome the token limit issue arising from a lengthy conversation history inherent to iterative programming, we selectively truncate the middle segment of the conversation history and resend the API call to aid in error recovery.

3.3.3 Safety and Quality Assurance Methods. During the development process, we identified several common mistakes made by the LLM during code generation [12]. These errors can be roughly categorized into two classes: **errors in interpretation** and **errors in execution** (see Appendix B for details). Errors in interpretation, such as import errors, are usually detected by the program interpreter or compiler, and are relatively straightforward to rectify. On the other hand, errors in execution, such as adding an unnecessary action, are less obvious because they do not necessarily lead to an immediate code breakdown. Instead, these errors become apparent only when undesirable task outcomes are observed. We explored various methods, such as parsing the generated code or adding rules to the initial prompt, to address errors during execution; grounded prompting was the most effective approach.

Alchemist relies on grounded prompting (*e.g.,* [54]) to reinforce specific rules for code output towards ensuring safety, quality and executability of the generated code. We have two general grounding prompts that are added to all prompts issued by the users: **(1)** "*By using the function library you are provided, "* is added to the beginning of each prompt to reinforce library usage and prevent LLM from using imaginary functions; and **(2)** " *make sure to move back to home after the task is finished.*" is added to the end of each prompt to ensure the robot's proper positioning between tasks.

In addition to these two groundings, we use three conditional groundings more specific to our exploratory study. If there is an "add" word in the user prompt, then we append the user input with " *Make sure to use marker location.*". If there is a "pour" word in the user prompt, then we add "*Don't move above the beaker before pouring; just call the pour function. Also, after pouring, make sure you place the object back to where it was on the table and then open the gripper to release it.*". If there are any of "function", "generic", "code" words in the user prompt, we add "*If you wrote a function, remember to add an example function call at the end.*". These groundings were all added on an as-needed basis. The first one is to make sure when objects are being added to the virtual space, GPT uses markers as location indicators; the second is to address several issues in its call of pour function and assumptions it makes; and the third is to make sure when generic functions are written by GPT there is also an accompanying example function call.

Other than prompt grounding, we added code verification. The primary objective of the code verification is to examine the code produced by GPT and make necessary general corrections such as imports (*e.g.,* not importing rospy, etc.), ROS node and function

library initialization, as well as Python version check. This verification ensures that the code created by GPT does not include errors resulting in failure of code interpretation.

3.4 System Modularity

3.4.1 Robotic Platform. Alchemist features modularized, interchangeable function library and initial prompts, allowing for the easy configuration to different robotic platforms with support for both physical and simulated robots and LLM used. In our initial system release, we provide implementations for two manipulators (Universal Robots UR5 and Franka Emika Panda) and one mobile manipulator (PAL Robotics TIAGo).

3.4.2 Vision System. The vision system in Alchemist offers an easy, fast and low cost way for quick testing and development. We modify the AR Track Alvar [34] package to have object grasping orientation information encoded in each marker. For this purpose, we created markers that have a pointy end indicating how the robot's gripper should approach and align itself with the marker inspired by Sefidgar et al. [47] (for details see supplementary materials). Importantly, these markers play a pivotal role in establishing a connection between physical objects and their virtual counterparts. They are intentionally designed to be independent of specific objects, offering users the flexibility to utilize them as reference points for object positioning. Users can strategically position these markers anywhere within the workspace, allowing for various orientations and placements. It is worth noting that Alchemist is engineered to support seamless integration of a full fledged perception system that has multi-modal sensing through its modular system architecture.

4 AN EXPLORATORY STUDY

We conducted an exploratory study to gauge the usability of our system and understand its limitations. Details of the experimental task and settings are provided in the supplementary materials [2].

4.1 Context and Task

Integrating robotic assistants in life sciences laboratories is emerging as a promising application domain [10, 11, 29]; the nature of lab experiments in life sciences requires precise and repetitive work with long operation duration and involves various health hazards. However, experts in life sciences rarely have experience programming or working with robots. Changes in an experimental protocol require robotics experts to set up or update the configuration of the robotic system to the new experimental requirements. We propose Alchemist as an alternative since its collaborative end-user programming paradigm is designed to help users with little to no robotics experience to be able to program robots to perform desired tasks. Thus, we chose a common biochemistry experiment as the basis for the programming task in our study.

We based our task on the *LB Media preparation* experiment, which is performed to create plates for growing live cultures in biochemistry and related field labs. We modified *LB Media preparation* task into a toy experiment by replacing reagents with beads and glass lab equipment such as beakers and graduated cylinders with plastic ones for safety reasons. At a high level, this task is about picking various graduated cylinders that have different types of reagents and pouring them into a beaker to create mixture.

In our main experimental task, we asked participants to use the system to generate general functions for re-usability but let them decide whether to use general functions or step by step prompting for the task execution. Before the main task, we used a training task to familiarize users with the system. Our training task was designed to give participants a complete overview of the system by requiring all system functionalities to be used.

4.2 Procedure

Participants were asked to fill out the consent form, and right after, they were given a user manual and asked to read it entirely. After reading the user manual, they were asked to watch a short tutorial video that explains the training task and the system. The experimenter then gave the participants the sheet containing the training task and went over it once to ensure the participants understood everything. Then, participants started the training task and the experimenter guided the participants along the tasks to familiarize them with the system. Upon completing the training task, the experimenter gave the participant the main task sheet, and started the video and screen recording. After that the experimenter went behind a divider to let participants work independently with the system. From this time till task end, the experimenter did not intervene unless there was a significant failure with the system usage or participants have spent more than 45 minutes on the task alone. If the experimenter stepped in, the experimenter was instructed to give just enough support to have participants go on with the task. A post-interaction questionnaire that collected System Usability Scale (SUS) [9] responses and demographic information including self-reported expertise in programming and robot programming was administered. After the questionnaire, participants had a semi-structured interview to conclude the study. The user manual, study procedure, interview questions, training task sheet, and main task sheet can be found in the supplementary materials.

4.3 Measures

We collected a range of metrics to understand the user experience of our system during the exploratory study (see Appendix C for details). Measures such as total programming time, debugging time, idle time, task completion time, number of errors, editor use, debugging method, and use of general functions are extracted using post-study labeling of the video and screen recordings of the participants interacting with the system and the robot during the study.

4.4 Participants

In this study, we recruited 5 (1 male, 4 female) graduate students or postdocs who work in biology, chemistry and biophysics fields ("novice"). Their age ranged from 25 to 29 ($M = 26.8, SD = 1.48$). We additionally recruited 5 (4 male, 1 female) graduate students who work in the robotics field. Their age ranged from 25 to 28 ($M = 26.4, SD = 1.34$). Participants who work in the robotics field are denoted as "experts" and provided us with additional insights about our system. Participants self-reported expertise levels were rated on a scale of 1 to 5 for coding and robot coding, with 1 indicating novice and 5 indicating expert. Novice users typically rated themselves

[2] http://tinyurl.com/supplementary-alchemist

Table 2: Reported measures from the exploratory study with respect to user expertise level

	PID	Prog. Time	Debugging Time	Idle Time	Task Comp. Time	# Errors	Error Types	Editor Use	Debugging Method	Use of Gen. Func.	SUS
Novice	N1	00:32:48	00:24:18	00:08:24	1:05:30	2	Name, Physical	no	prompt	Yes	72.5
	N2	00:09:01	00:00:00	00:26:11	0:35:12	0		no	-	No	52.5
	N3	00:13:24	00:01:20	00:59:28	1:14:12	1	Syntax	no	prompt	No	72.5
	N4	00:44:35	00:32:06	00:14:30	1:31:11	5	Name,Factual,Syntax,Import	no	prompt	No	12.5
	N5	00:15:57	00:10:35	00:22:33	0:49:05	1	Factual	yes	prompt	Yes	70
Mean		00:23:09	00:13:40	00:26:13	1:03:02	1.8					56
Expert	E1	00:20:09	00:13:25	00:33:12	0:54:46	2	Factual,Syntax	yes	edit code	Yes	70
	E2	00:32:29	00:14:21	01:14:17	2:01:07	1	Name	yes	edit code	No	62.5
	E3	00:14:00	00:04:02	00:50:57	0:56:59	5	Name	yes	edit code	Yes	72.5
	E4	00:06:46	00:00:00	00:34:36	0:41:22	0		yes	edit code	Yes	80
	E5	00:10:01	00:01:24	00:24:15	0:35:40	1	Name	no	prompt	Yes	57.5
Mean		00:16:41	00:06:38	00:43:27	1:01:59	1.8					68.5

as 2 for coding ($M = 2.42, SD = 1.51$) and 1 for robot coding ($M = 1.71, SD = 1.25$), while expert users rated themselves as 4 for coding ($M = 4.25, SD = 0.5$) and 4 for robot coding ($M = 4, SD = 0.82$).

4.5 Findings

From our exploratory study we observed some interesting similarities and differences in how novice and expert users interacted with the system (Table 2). Below, we highlight these observations.

Novices and experts both had similar average task completion times which might be due to different prior knowledge of both groups, one group possessing prior knowledge about the task well whereas the other group possessing prior knowledge on coding and robots. This is supported by how the expert group on average took less time programming and debugging but more idle time than the novice group; idle time refers to time spent on the task other than using the system (*e.g.,* thinking, reading the user manual, etc.).

Novice users tended to debug their program by prompting the LLM further rather than using the editor; this behavior can be attributed to novice users avoiding directly dealing with code due to their unfamiliarity as reflected by **N2**: "*It would be difficult for me to troubleshoot by myself, as I lack the confidence to examine the code to determine exactly what's going on.*". Similarly, we observed that novice participants chose not to use general functions; **N3**: "*Since I'm not comfortable with coding, I prefer providing direct step-by-step instructions to the robot rather than using a generic function.*" A prompt example by novice **N3** illustrates this observation: "*pour the water from the 250ml graduated cylinder to the 500ml beaker. Put the 250ml graduated cylinder back to its original place gently.*" On the other hand, a prompt example by expert **E1** for the same task shows that experts tended to create generic functions and reused them: "*Can you write a generic function called "pick_and_pour", which allows me to insert the input, and the robot will based on the input to grab the target and pour it into the 500mL beaker?*". We provide more examples of user prompts in supplementary materials of this paper.

Overall, all novice participants had positive comments about the potential of the collaborative programming paradigm in democratizing robot programming especially in specialized domains such as life sciences research laboratories. **N3** commented that "*Working in a biochemistry lab, we need to prepare media almost every day, often in large quantities. It's a routine protocol that takes up a significant amount of time. However, if a system like this could be used to automate the process, it could save a substantial amount of time.*". **N5** also commented that "*I think the idea behind this project is really novel. It can be used extensively, especially in biochemistry labs, where tasks like these are a frequent occurrence. I think this system could be extremely helpful.*". This feedback validates our rationale behind developing Alchemist: empowering true end-users to intuitively create, edit, and test robot programs for custom uses.

The main limitations of the system arose due to problems with motion planning and visual perception resulting in failures in robot action which occurred independent of the generated code itself; our findings highlight the value of this collaborative programming paradigm while underscoring the need to improve reliability in task execution. Participant N4 experienced a significant number of vision and planning errors, which explains the outlier SUS score of the participant; **N4**: "*I don't feel confident using this in a lab because it keeps knocking over things several times.*" . Yet, N4 still acknowledged the value of a system like Alchemist: "*I would love to have a liquid handling system in our lab where I could simply press a button and say 'go' without worrying about it failing, but I don't think it's at that stage yet.*".

We note that although the occurrence of errors described in Appendix B has decreased due to our code verification mechanisms, errors have not been completely eliminated.

5 LESSONS LEARNED

Our system design and development process, along with our observations in the exploratory study, elucidated a range of effective practices for and limitations of our end-user programming approach. We are sharing a set of lessons learned to guide future work on LLM-powered end-user robot programming.

5.1 LLMs Can Output Unreliable Code

Robustness of LLM-generated code is critical for a successful end-user programming experience [30, 50]. We adopted two key strategies to enhance the reliability of the generated programs: **(1) code verification** through simple parsing and error handling mechanisms helped eliminate most interpretation errors; and **(2) grounded**

prompting helped reduce execution errors and refine LLM responses to fit domain-specific requirements. These approaches helped minimize errors ensuring that even novice users were able to successfully complete our study task; yet, some errors still persisted introducing overhead in the task completion time by necessitating debugging (see Table 2). Noticing that novice users largely relied on prompting the LLM for debugging further underscored the need for reliable code generation and iterative refinement to enable end-users to effectively author robot programs. Incorporating advanced formal software verification methods [32, 52] into LLM-enabled end-user programming systems can further bolster the reliability of LLM-generated programs and enhance the efficiency of our proposed approach.

> **Lesson Learned**: *Enhancing LLM-generated code reliability through code verification and effective prompting is critical for end-user robot programming.*

5.2 Effective LLM Prompting is Difficult

A significant challenge stems from users possessing a skewed or incomplete understanding of the LLM and its capabilities [6, 31, 57]. This often leads to vague or implicit prompts, which in turn can produce undesirable code outcomes, ranging from coding errors to unintended robot behaviors. To address this issue, we implemented two strategies to improve how users prompt the LLM.

Firstly, we used guided training assets (*i.e.,* user manual, tutorial video, a training task) in our exploratory study to better familiarize the users with methods for LLM prompting. However, effective prompting of large language models remains a challenge for users and further work is needed to develop training methods that empower end-users to create high-quality prompts.

Secondly, we also utilized grounded prompting to dynamically add contextual details to user prompts, *e.g.,* when users prompted the LLM about a pouring task we added grounding to the prompt to state that the container once poured should be returned to its original location. However, this mechanism is domain-dependent and requires prompt modification in case of a vastly different application domain reflecting a limitation of this approach. One potential solution could be the structural integration of the LLM within the robot programming ecosystem to ensure the LLM has necessary contextual information. Moreover, enhancing dialogue between the LLM and the user about the task's overarching purpose and characteristics could strengthen the collaborative bond and enrich the LLM's contextual comprehension. These approached could facilitate the automatic grounding of tasks, based on current requirements and conditions.

> **Lesson Learned**: *Effective LLM prompting requires end-user training and dynamic context-dependent prompt enhancement.*

5.3 End-User Aversion to Direct Coding

End-users exhibit a wide range of knowledge regarding programming and specifically robot programming. To cater to this diversity, Alchemist features a two-level abstraction in its function library which affects how the LLM generates code. The high-level abstraction functions aimed to guide the LLM to generate less error prone and easier to debug programs while the low-level abstraction provided the end-user with more control over program specification. This distinction was important for novice users especially as we observed they tend to avoid directly viewing and editing code.

LLMs are capable of generating general purpose code that can be reusable in later stages of a task. Alchemist provides users with a terminal panel function (see Appendix A for details) that allows them to call general functions with custom inputs without editing any code; however, we still observed that novice users hesitated to use general functions. Further integrating LLMs into the programming ecosystem as a conversational assistant could allow novice users to call such functions without needing to using the command line interface, thus further enhancing a sense of collaboration. Further work is needed to design methods to empower end-users to effectively use more advanced programming notions (*e.g.,* general functions) to maximally exploit the generative capabilities of LLMs.

> **Lesson Learned**: *Introducing abstractions to minimize code complexities while retaining programmatic expressiveness can enhance user confidence in programming.*

6 LIMITATIONS AND FUTURE WORK

Though our exploratory study provided an insight into the effectiveness of Alchemist as well as how expert and novice users interacted with it, the sample size is small, limiting our ability to draw conclusions regarding how people with different levels of programming knowledge may use an LLM-based robot programming system; moreover, the stochastic nature of LLM outputs may result in various programming/user behavior and user experience [37]. Future work should investigate methods to validate LLM-based robot programming and conduct well-powered experiments. Furthermore, while the inclusion of expert users in our exploratory study offered valuable insights into how experts and novices differ in using the system, it limited our ability to focus the study on true end-users.

While true end-users (*i.e.,* participants from life sciences) evaluated our system, their interactions occurred in a robotics laboratory, rather than a real-world deployment environment which represents another limitation. Additionally, this paper centers on a singular use case for our system. However, its potential applications span diverse settings, from varying experimental protocols to entirely distinct domains like manufacturing or customer-facing services in hospitality and food sectors. Future research should explore deploying this system or analogous ones in real-world contexts across various domains to discern both its capabilities and constraints. Lastly, future research should systematically compare our end-user programming system with existing state-of-the-art systems to better evaluate and understand their differences.

In conclusion, we see a shift toward a more collaborative paradigm for end-user robot programming as powered by large language models and vast opportunities to use LLMs in advancing end-user development of robot applications in diverse domains.

ACKNOWLEDGMENTS

This work was supported by National Science Foundation award #2143704.

REFERENCES

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, et al. 2022. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691* (2022).

[2] Gopika Ajaykumar, Maureen Steele, and Chien-Ming Huang. 2021. A Survey on End-User Robot Programming. *ACM Comput. Surv.* 54, 8, Article 164 (oct 2021), 36 pages. https://doi.org/10.1145/3466819

[3] Baris Akgun, Maya Cakmak, Jae Wook Yoo, and Andrea L. Thomaz. 2012. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *2012 7th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. 391–398. https://doi.org/10.1145/2157689.2157815

[4] Pierre A. Akiki, Paul A. Akiki, Arosha K. Bandara, and Yijun Yu. 2020. EUD-MARS: End-user development of model-driven adaptive robotics software systems. *Science of Computer Programming* 200 (2020), 102534. https://doi.org/10.1016/j.scico.2020.102534

[5] Sonya Alexandrova, Zachary Tatlock, and Maya Cakmak. 2015. RoboFlow: A flow-based visual programming language for mobile manipulation tasks. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 5537–5544. https://doi.org/10.1109/ICRA.2015.7139973

[6] Victor Nikhil Antony and Chien-Ming Huang. 2023. ID. 8: Co-Creating Visual Stories with Generative AI. *arXiv preprint arXiv:2309.14228* (2023).

[7] Emilia I Barakova, Jan CC Gillesen, Bibi EBM Huskens, and Tino Lourens. 2013. End-user programming architecture facilitates the uptake of robots in social therapies. *Robotics and Autonomous Systems* 61, 7 (2013), 704–713.

[8] Ankica Barišić, João Cambeiro, Vasco Amaral, Miguel Goulão, and Tarquínio Mota. 2018. Leveraging teenagers feedback in the development of a domain-specific language: the case of programming low-cost robots. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*. 1221–1229.

[9] John Brooke. 1995. SUS: A quick and dirty usability scale. *Usability Eval. Ind.* 189 (11 1995).

[10] Benjamin Burger, Phillip M. Maffettone, Vladimir V. Gusev, Catherine M. Aitchison, Yang Bai, Xiaoyan Wang, Xiaobo Li, Ben M. Alston, Buyi Li, Rob Clowes, Nicola Rankin, Brandon Harris, Reiner Sebastian Sprick, and Andrew I. Cooper. 2020. A mobile robotic chemist. *Nature* 583, 7815 (01 Jul 2020), 237–241. https://doi.org/10.1038/s41586-020-2442-2

[11] Chih-Lin Chen, Ting-Ru Chen, Shih-Hao Chiu, and Pawel L. Urban. 2017. Dual robotic arm "production line" mass spectrometry assay guided by multiple Arduino-type microcontrollers. *Sensors and Actuators B: Chemical* 239 (2017), 608–616. https://doi.org/10.1016/j.snb.2016.08.031

[12] Juo-Tung Chen and Chien-Ming Huang. 2023. Forgetful Large Language Models: Lessons Learned from Using LLMs in Robot Programming. *arXiv preprint arXiv:2310.06646* (2023).

[13] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374* (2021).

[14] Bruno Pereira Cipriano and Pedro Alves. 2023. GPT-3 vs Object Oriented Programming Assignments: An Experience Report. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 61–67. https://doi.org/10.1145/3587102.3588814

[15] Enrique Coronado, Dominique Deuff, Pamela Carreno-Medrano, Leimin Tian, Dana Kulić, Shanti Sumartojo, Fulvio Mastrogiovanni, and Gentiane Venture. 2021. Towards a Modular and Distributed End-User Development Framework for Human-Robot Interaction. *IEEE Access* 9 (2021), 12675–12692. https://doi.org/10.1109/ACCESS.2021.3051605

[16] Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. 2023. Task and motion planning with large language models for object rearrangement. *arXiv preprint arXiv:2303.06247* (2023).

[17] GitHub Documentation. 2023. GitHub copilot documentation. https://docs.github.com/en/copilot

[18] Danny Driess, Fei Xia, Mehdi SM Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, et al. 2023. Palm-e: An embodied multimodal language model. *arXiv preprint arXiv:2303.03378* (2023).

[19] Floris Erich, Masakazu Hirokawa, and Kenji Suzuki. 2017. A visual environment for reactive robot programming of macro-level behaviors. In *Social Robotics: 9th International Conference, ICSR 2017, Tsukuba, Japan, November 22-24, 2017, Proceedings 9*. Springer, 577–586.

[20] Daniela Fogli, Luigi Gargioni, Giovanni Guida, and Fabio Tampalini. 2022. A hybrid approach to user-oriented programming of collaborative robots. *Robotics and Computer-Integrated Manufacturing* 73 (2022), 102234. https://doi.org/10.1016/j.rcim.2021.102234

[21] Yuxiang Gao and Chien-Ming Huang. 2019. PATI: a projection-based augmented table-top interface for robot programming. In *Proceedings of the 24th international conference on intelligent user interfaces*. 345–355.

[22] Jayanto Halim, Paul Eichler, Sebastian Krusche, Mohamad Bdiwi, and Steffen Ihlenfeldt. 2022. No-Code robotic programming for agile production: A new markerless-approach for multimodal natural interaction in a human-robot collaboration context. *Frontiers in Robotics and AI* 9 (2022), 1001955.

[23] Justin Huang and Maya Cakmak. 2017. Code3: A System for End-to-End Programming of Mobile Manipulator Robots for Novices and Experts. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction* (Vienna, Austria) *(HRI '17)*. Association for Computing Machinery, New York, NY, USA, 453–462. https://doi.org/10.1145/2909824.3020215

[24] Justin Huang, Dieter Fox, and Maya Cakmak. 2019. Synthesizing Robot Manipulation Programs from a Single Observed Human Demonstration. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 4585–4592. https://doi.org/10.1109/IROS40897.2019.8968543

[25] Wenlong Huang, F. Xia, Ted Xiao, Harris Chan, Jacky Liang, Peter R. Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *Conference on Robot Learning*. https://api.semanticscholar.org/CorpusID:250451569

[26] Takashi Inagaki, Akari Kato, Koichi Takahashi, Haruka Ozaki, and Genki N Kanda. 2023. LLMs can generate robotic scripts from goal-oriented instructions in biological laboratory automation. *arXiv preprint arXiv:2304.10267* (2023).

[27] Michal Kapinus, Vítězslav Beran, Zdeněk Materna, and Daniel Bambušek. 2019. Spatially situated end-user robot programming in augmented reality. In *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 1–8.

[28] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 9493–9500.

[29] Hui Liu, Norbert Stoll, Steffen Junginger, and Kerstin Thurow. 2014. A Fast Approach to Arm Blind Grasping and Placing for Mobile Robot Transportation in Laboratories. *International Journal of Advanced Robotic Systems* 11, 3 (2014), 43. https://doi.org/10.5772/58253 arXiv:https://doi.org/10.5772/58253

[30] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *arXiv preprint arXiv:2305.01210* (2023).

[31] Michael Xieyang Liu, Advait Sarkar, Carina Negreanu, Benjamin Zorn, Jack Williams, Neil Toronto, and Andrew D Gordon. 2023. "What It Wants Me To Say": Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–31.

[32] Matt Luckcuck, Marie Farrell, Louise A Dennis, Clare Dixon, and Michael Fisher. 2019. Formal specification and verification of autonomous robotic systems: A survey. *ACM Computing Surveys (CSUR)* 52, 5 (2019), 1–41.

[33] Marco Manca, Fabio Paternò, and Carmen Santoro. 2022. End-user development in industrial contexts: the paper mill case study. *Behaviour and Information Technology* 41, 9 (July 2022), 1848–1864. https://doi.org/10.1080/0144929X.2022.208

[34] Scott Niekum. 2016. AR Track Alvar. http://wiki.ros.org/ar_track_alvar

[35] Soh-Khim Ong, AWW Yew, Naresh Kumar Thanigaivel, and Andrew YC Nee. 2020. Augmented reality-assisted robot programming system for industrial applications. *Robotics and Computer-Integrated Manufacturing* 61 (2020), 101820.

[36] OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL]

[37] Shuyin Ouyang, Jie M Zhang, Mark Harman, and Meng Wang. 2023. LLM is Like a Box of Chocolates: the Non-determinism of ChatGPT in Code Generation. *arXiv preprint arXiv:2308.02828* (2023).

[38] Chris Paxton, Andrew Hundt, Felix Jonathan, Kelleher Guerin, and Gregory D. Hager. 2016. CoSTAR: Instructing Collaborative Robots with Behavior Trees and Vision. arXiv:1611.06145 [cs.RO]

[39] Chris Paxton, Felix Jonathan, Andrew Hundt, Bilge Mutlu, and Gregory D Hager. 2018. Evaluating methods for end-user creation of robot task plans. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 6086–6092.

[40] David Porfirio, Laura Stegner, Maya Cakmak, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2023. Sketching Robot Programs On the Fly. In *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction* (Stockholm, Sweden) *(HRI '23)*. Association for Computing Machinery, New York, NY, USA, 584–593. https://doi.org/10.1145/3568162.3576991

[41] David J. Porfirio, Laura Stegner, Maya Cakmak, Allison Sauppé, Aws Albarghouthi, and Bilge Mutlu. 2021. Figaro: A Tabletop Authoring Environment for Human-Robot Interaction. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (Yokohama, Japan) *(CHI '21)*. Association for Computing Machinery, New York, NY, USA, Article 414, 15 pages. https://doi.org/10.1145/3411764.3446864

[42] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. 2022. Robust Speech Recognition via Large-Scale Weak Supervision. arXiv:2212.04356 [eess.AS]

[43] Nico Ritschel, Felipe Fronchetti, Reid Holmes, Ronald Garcia, and David C. Shepherd. 2022. Can Guided Decomposition Help End-Users Write Larger Block-Based Programs? A Mobile Robot Experiment. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 133 (oct 2022), 26 pages. https://doi.org/10.1145/3563296

[44] Steven I. Ross, Fernando Martinez, Stephanie Houde, Michael Muller, and Justin D. Weisz. 2023. The Programmer's Assistant: Conversational Interaction with a Large Language Model for Software Development. In *Proceedings of the 28th International Conference on Intelligent User Interfaces* (Sydney, NSW, Australia) *(IUI '23)*. Association for Computing Machinery, New York, NY, USA, 491–514. https://doi.org/10.1145/3581641.3584037

[45] Gregory F Rossano, Carlos Martinez, Mikael Hedelind, Steve Murphy, and Thomas A Fuhlbrigge. 2013. Easy robot programming concepts: An industrial perspective. In *2013 IEEE international conference on automation science and engineering (CASE)*. IEEE, 1119–1126.

[46] Jaromir Savelka, Arav Agarwal, Christopher Bogart, Yifan Song, and Majd Sakr. 2023. Can Generative Pre-trained Transformers (GPT) Pass Assessments in Higher Education Programming Courses? *arXiv preprint arXiv:2303.09325* (2023).

[47] Yasaman S. Sefidgar, Prerna Agarwal, and Maya Cakmak. 2017. Situated Tangible Robot Programming. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*. 473–482.

[48] Sabarathinam Shanmugam, Anjana Hari, Ashok Pandey, Thangavel Mathimani, LewisOscar Felix, and Arivalagan Pugazhendhi. 2020. Comprehensive review on the application of inorganic and organic nanoparticles for enhancing biohydrogen production. *Fuel* 270 (2020), 117453. https://doi.org/10.1016/j.fuel.2020.117453

[49] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 11523–11530.

[50] Priyan Vaithilingam, Tianyi Zhang, and Elena L. Glassman. 2022. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, Article 332, 7 pages. https://doi.org/10.1145/3491101.3519665

[51] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2023. *ChatGPT for Robotics: Design Principles and Model Abilities*. Technical Report MSR-TR-2023-8. Microsoft. https://www.microsoft.com/en-us/research/publication/chatgpt-for-robotics-design-principles-and-model-abilities/

[52] Dennis Walter, Holger Täubig, and Christoph Lüth. 2010. Experiences in applying formal verification in robotics. In *International Conference on Computer Safety, Reliability, and Security*. Springer, 347–360.

[53] David Weintrop, Afsoon Afzal, Jean Salac, Patrick Francis, Boyang Li, David C Shepherd, and Diana Franklin. 2018. Evaluating CoBlox: A comparative study of robotics programming environments for adult novices. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. 1–12.

[54] Orion Weller, Marc Marone, Nathaniel Weir, Dawn Lawrie, Daniel Khashabi, and Benjamin Van Durme. 2023. "According to ..." Prompting Language Models Improves Quoting from Pre-Training Data. arXiv:2305.13252 [cs.CL]

[55] Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas A. Funkhouser. 2023. TidyBot: Personalized Robot Assistance with Large Language Models. *ArXiv* abs/2305.05658 (2023). https://api.semanticscholar.org/CorpusID:258564887

[56] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. 2023. Language to Rewards for Robotic Skill Synthesis. *arXiv preprint arXiv:2306.08647* (2023).

[57] JD Zamfirescu-Pereira, Richmond Y Wong, Bjoern Hartmann, and Qian Yang. 2023. Why Johnny can't prompt: how non-AI experts try (and fail) to design LLM prompts. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–21.

[58] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. 2023. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592* (2023).

[59] Helmut Zörrer, Georg Weichhart, Mathias Schmoigl Tonis, Till Bieg, Matthias Propst, Dominik Schuster, Nadine Sturm, Chloé Nativel, Gabriele Salomon, Felix Strohmeier, Andreas Sackl, Michael Eberle, and Andreas Pichler. 2023. Enabling End-Users in Designing and Executing of Complex, Collaborative Robotic Processes. *Applied System Innovation* 6, 3 (2023). https://doi.org/10.3390/asi6030056