5 – RT-Grasp: Reasoning Tuning Robotic Grasping via Multi-modal Large Language Model

The growth of artificial intelligence in recent years has been significantly driven by the emergence of large language models (LLMs). These models, packed with vast knowledge and advanced reasoning ability, have revolutionized our approach to various tasks, especially those involving language processing. In robotics, LLMs play a crucial role in facilitating direct interactions between robots and humans. For instance, in tasks such as robot manipulation planning, many studies [1], [2], [3] have utilized LLMs to interpret natural language commands from users and translate them into feasible multi-step plans for robots. However, despite their potential in robotics, LLMs' application has predominantly been limited to such planning tasks. A notable bottleneck lies in the textual nature of LLM outputs, which often pose challenges for tasks requiring precise numerical outputs. Recently, multi-modal LLMs have expanded LLM capabilities by understanding both text and images. In robotics, they bridge the gap between perception and planning, addressing a variety of embodied reasoning tasks [4], [5]. However, their image understanding lacks precision, for

In summary, our work focuses on adapting multi-modal LLMs for numerical prediction tasks, specifically in the domain of robotic grasping. In contrast to deterministic traditional methods, our approach not only incorporates advanced reasoning capabilities but also introduces a novel paradigm for refining predictions, as illustrated in Fig. 1. The main contributions can be summarized as follows

Traditional robotic grasping methods typically rely on deterministic predictions, which often fail in real-world scenarios due to their lack of reasoning capabilities. Most existing methods [7], [8], using CNN-based architectures, excel in experimental accuracy on benchmark datasets, but struggle in practical applications. For example, these traditional models may produce theoretically correct predictions that prove impractical in execution, as shown in Fig. 1 and labeled as invalid. Such predictions are hard to apply across robot arms due to varying gripper constraints. Additionally, some theoretically correct grasps may result in unsafe actions, such as targeting the sharp ends of screwdrivers during grasping. Hence, adopting a non-deterministic approach equipped with reasoning ability is crucial. This capability not only allows the model to generate practical grasp poses applicable across various settings but also allows the refinement of predictions based on user commands. Here a question is posed: can the reasoning capabilities inherent in LLMs be utilized for numerical prediction tasks in robotics? This paper offers a positive answer, showcasing an adaptation of multi-modal LLMs to robotic grasping tasks

IV. RT-GRASP In this section, we introduce Reasoning Tuning for robotic grasping (RT-Grasp), a novel method designed to bridge the gap between the inherent text-centric nature of LLMs and the precise numerical requirements of robotic tasks. Its primary objective is to facilitate multi-modal LLMs for numerical prediction by leveraging their extensive encapsulated prior knowledge. A pre-trained multi-model LLM, such as LLaVA [37], can be directly fine-tuned in a fully supervised manner when given the image and the text instruction. The model is trained by predicting each token in the text output sequentially. The proposed Reasoning Tuning introduces a structured text output, which includes a reasoning phase and a subsequent numerical prediction. We created our image-text dataset for robotic grasping, named Reasoning Tuning VLM (Visual Language Model) Grasp dataset, used for fine-tuning multimodal LLMs. Additionally, we introduce a method that automatically generates such image-text datasets using GPT3.5 [38], which can be applied to datasets for tasks beyond robotic grasping. Further details are presented in Section IV-A. Furthermore, we discuss two cost-efficient training strategies employed in our experiments in Section IV-B reasoning_templates = { [“The image shows a baseball, which is spherical. Grasping it as its center will ensure optimal balance.” , “The object is a baseball. Its round shape requires a center grip for stability.” , …] , [“The object is a pair of sunglasses, with lenses and a frame. Grasping the frame, away from the lenses, is safest.” , “Recognized sunglasses. It's essential to grasp its frame while avoiding the delicate lenses.” , …] , “baseball”: “sunglasses”: [“The object is a cup, which is generally cylindrical. An secure way to grasp a cup is by its edge from the top and the rotation angle should be such that the gripper is orthogonal to the edge of the cup.” , “Recognized object is a cup. Its best grasped by targeting its top edge, ensuring the gripper is perpendicular to its circular opening.” , …] , “cup”: … } “object category”: [“Description of the object type and general shape. Grasping strategy.”, …] Fig. 4. Examples of reasoning templates within the Reasoning Tuning VLM dataset.

This research underscores the potential of LLMs beyond their conventional text-centric applications. Our proposed method utilizes the extensive prior knowledge of LLMs for numerical predictions, specifically in robotic grasping. Through comprehensive experiments conducted on both benchmark datasets and real-world scenarios, we have demonstrated the efficacy of our approach. For future work, we plan to extend the validation of our method by applying it to grasping datasets featuring a broader array of objects, such as the Jacquard dataset [42]. Moreover, the adaptation of multi-modal LLMs for numerical predictions in other robotic manipulation tasks is also a promising research direction

6 – ARRC: Advanced Reasoning Robot Control—Knowledge-Driven Autonomous Manipulation Using Retrieval-Augmented Generation

Abstract— We present ARRC (advanced reasoning robot control), a practical system that connects natural language instructions to safe, local robotic control by combining RetrievalAugmented Generation (RAG) with RGB–D perception and guarded execution on an affordable robot arm. The system indexes curated robot knowledge (movement patterns, task templates, and safety heuristics) in a vector database, retrieves task-relevant context for each instruction, and conditions a large language model (LLM) to synthesize JSON-structured action plans. These plans are executed on a UFactory xArm 850 fitted with a Dynamixel-driven parallel gripper and an Intel RealSense D435 camera. Perception uses AprilTags detections fused with depth to produce object-centric metric poses; execution is enforced via a set of software safety gates (workspace bounds, speed/force caps, timeouts, and bounded retries). We describe the architecture, knowledge design, integration choices, and a reproducible evaluation protocol for tabletop scan/approach/pick–place tasks. Experimental results are reported to demonstrate efficacy of the proposed approach. Our design shows that RAG-based planning can substantially improve plan validity and adaptability while keeping perception and low-level control local to the robot.

Abstract— We present ARRC (advanced reasoning robot control), a practical system that connects natural language instructions to safe, local robotic control by combining RetrievalAugmented Generation (RAG) with RGB–D perception and guarded execution on an affordable robot arm. The system indexes curated robot knowledge (movement patterns, task templates, and safety heuristics) in a vector database, retrieves task-relevant context for each instruction, and conditions a large language model (LLM) to synthesize JSON-structured action plans. These plans are executed on a UFactory xArm 850 fitted with a Dynamixel-driven parallel gripper and an Intel RealSense D435 camera. Perception uses AprilTags detections fused with depth to produce object-centric metric poses; execution is enforced via a set of software safety gates (workspace bounds, speed/force caps, timeouts, and bounded retries). We

describe the architecture, knowledge design, integration choices, and a reproducible evaluation protocol for tabletop scan/approach/pick–place tasks. Experimental results are reported to demonstrate efficacy of the proposed approach. Our design shows that RAG-based planning can substantially improve plan validity and adaptability while keeping perception and low-level control local to the robot.

In this paper, we bridge these gaps by introducing a RAG-enabled robotic manipulation pipeline–called ARRC (advanced reasoning robot control)–that unifies perception, retrieval, and safe plan execution. We deploy this system on a UFactory xArm 850 equipped with a RealSense D435 and a Dynamixel gripper, integrating retrieval of robot-centric safety heuristics and procedural templates at inference time. This design offers both adaptability and reliability, enabling the injection of new task knowledge or safety rules without retraining. Our contributions are as follows: • We develop a hybrid VLA architecture integrating RAG for dynamic injection of procedural and safety knowledge. • We demonstrate a real-world implementation with RGB-D perception, JSON-structured plan generation, and strict execution safety gates. • We propose a reproducible ablation protocol to quantify contributions of retrieval, vision gating, and safety checks.

II. SYSTEM OVERVIEW A. Perception AprilTags, combined with depth data from the Intel RealSense D435, provide marker-based detections that are fused to recover metric 3D poses in the robot frame. With sub-pixel accuracy, AprilTags enable robust and precise pose estimation, making them well-suited for manipulation tasks [19]. B. Retrieval & Planning We construct a curated robotics knowledge base comprising movement primitives, templates, safety heuristics, short demonstration transcripts, and parameterized affordances. This knowledge base is embedded using a SentenceTransformers model and indexed in ChromaDB (or a FAISS index). At inference time, the retrieval module selects the topk relevant context snippets, which are concatenated with the current observation summary and provided to an LLM (e.g., Gemini, PaLM-E-style models). The LLM then generates a structured JSON plan, enabling downstream execution [20]. C. Execution The JSON plan—represented as a sequence of named actions with bounded parameters—is first validated by a plan checker, synchronized with the latest object observations, and then executed through the XArm Python SDK. Execution is safeguarded by software safety gates, including workspace and joint limits, capped Cartesian speeds and accelerations, gripper torque/time gating, per-step timeouts, and bounded retries. Low-level controllers and perception remain local to the robot, while the LLM planner can be configured to run either locally (on-device) or via cloud APIs. IV. PROPOSED METHOD In this section, we detail each system component and our design choices.

D. Plan Representation, Validation, and Safety Plans follow a restricted JSON schema: { "goal": "place bottle on tray", "steps": [{"action":"SCAN_AREA", "params":{...}}, {"action":"APPROACH_OBJECT", "params":{"label":"bottle", "hover_mm":40}}, {"action":"MOVE_TO_POSE", "params":{"xyz_mm":[...], "rpy_deg":[...]}}] } Before execution, each action step is validated against system constraints, including parameter bounds such as speeds and positions, and overall plan length. Steps that require current environmental information are synchronized with the latest perception data, and high-risk operations may optionally include human-in-the-loop confirmation. During execution, the executor enforces runtime safety through mechanisms such as per-step timeouts, gripper aborts triggered by load or duration limits, and emergency retreat in response to repeated failures.

We have presented a practical, retrieval-augmented manipulation system that integrates local RGB–D perception, a vector-indexed robotics knowledge base, and LLM-driven plan synthesis to translate natural language instructions into validated, executable actions on a UFactory xArm 850. Experimental results demonstrate that retrieval-augmented generation improves plan specificity by providing concise, robot-relevant context, enabling more reliable and parameterized plans compared to LLM-only approaches. Furthermore, keeping perception and low-level control local proves effective for real-time execution and safety enforcement, while latency introduced by retrieval and LLM inference can be mitigated through capturing strategies. Despite these strengths, the current system has some limitations. The knowledge base is curated and static, limiting adaptability to lifelong updates. Tasks requiring tactile feedback or precise torque control, such as dowel insertion or screw driving, remain out of scope. Additionally, the LLM planner may occasionally generate physically infeasible actions, highlighting the necessity for robust plan validation and symbolic feasibility checks. These limitations underscore important avenues for future research. Future work will focus on scaling the system to more complex manipulation scenarios, including multi-arm coordination, large-scale benchmarking across diverse objects and lighting conditions, and integration of tactile sensing and force-aware motion primitives. On-device acceleration of retrieval and LLM inference is another promising direction to reduce latency and improve autonomy. Overall, our study demonstrates that combining retrieval-augmented language reasoning with local perception and safety-guarded execution provides a practical and reproducible pathway toward robust, adaptable robotic manipulation in real-world environments

D. Safety-Constrained Execution The execution module enforces a set of safety constraints to guarantee reliable operation during plan execution. First, all commanded target positions are validated against the robot's reachable workspace, ensuring all $P_a \in W$. Motion commands are further regulated through velocity limits: Cartesian translations satisfy $\|\dot{P}\| \leq v_{max}$, while joint velocities are constrained by $|\dot{\theta}_i| \leq \dot{\theta}_{i,max}$. Each action step is assigned a maximum allowable duration t_{max}; exceeding this limit triggers an abort and initiates a return-to-safe configuration. Gripper operations are protected through realtime load monitoring, with an emergency release executed if excessive force or unexpected obstruction is detected. By integrating workspace validation, dynamic constraints, temporal bounds, and force-aware gripper control, the execution framework achieves robust, safe deployment of RAGgenerated plans without sacrificing real-time responsiveness or hardware longevity.

7 - ROS-LLM: A ROS framework for embodied AI with task feedback and structured reasoning

Abstract We present a framework for intuitive robot programming by non-experts, leveraging natural language prompts and contextual information from the Robot Operating System (ROS). Our system integrates large language models (LLMs), enabling non-experts to articulate task requirements to the system through a chat interface. Key features of the framework include: integration of ROS with an AI agent connected to a plethora of open-source and commercial LLMs, automatic extraction of a behavior from the LLM output and execution of ROS actions/services, support for three behavior modes (sequence, behavior tree, state machine), imitation learning for adding new robot actions to the library of possible actions, and LLM reflection via human and environment feedback. Extensive experiments validate the framework, showcasing robustness, scalability, and versatility in diverse scenarios, including long-horizon tasks, tabletop rearrangements, and remote supervisory control. To facilitate the adoption of our framework and support the reproduction of our results, we have made our code open-source. You can access it at: ROS-LLM-Code.

As we conceptualize more tasks, we subsequently develop more and more atomic actions that we can curate in a library. Each atomic action is ultimately a function: an input variable and parameters that are mapped to a return value. The representation of an atomic action could be simple, for instance, “open gripper” that sends an “open” signal to a parallel gripper attached to a robot arm which then returns the final width from the gripper. The representation could take other forms, for example a planner and feedback controller, a neural network policy trained using RL or imitation learning for the end-effector that feeds into an inverse-kinematic controller to compute target joint states. The planner/controller parameters for a given formulation or the neural network weights can be stored in memory and considered as the atomic action representation in the atomic action library. This library of atomic actions can be in the form of a code API or, if the Robot Operating System (ROS) is being used, the library can be a list of ROS actions and services. Textual descriptions for each atomic action are always assumed to be provided, i.e. documentation.

In general, robot systems are designed in a modular way to allow users to build their own frameworks by easily integrating and modifying existing processes. The most well-known framework used in research and industry is the Robot Operating System (ROS) [41, 32]. Other examples include the Lightweight Communications and Marshaling (LCM) project [16] and the Open Robot Control Software (Orocos) project [5]. The ROS framework provides a well-established ecosystem of packages and libraries that are ready to use and integrated with many common robot systems (e.g. Universal Robots, Robotiq, Clearpath Robotics). Some of most widely used ROS libraries and packages include the TF library [11], MoveIt [9], and the Navigation-stack. The ROS ecosystem of contributors additionally contains many packages for many important requirements such as simulation [26, 34], kinematic modeling [47, 2], and planning and control [19, 36, 8]. Thus, ROS offers many packages providing useful functionalities for both research and commercial applications. These packages include valuable data structures, control interfaces, inverse kinematics (IK) and motion planning tools, perception utilities, and various visualizers. Additionally, with new tools such as the BehaviorTree.ROS library, ROS actions and services enable the generalization of a wide variety of capabilities required by robot systems into a unified execution framework.

As described in the introduction, we can think of an atomic action as a single task that the system can perform. Formally, we frame a single task as a Markov Decision Process (MDP) characterized by the tuple $\langle S, A, r,P, \gamma\rangle$ where S is the state space, A is the action space, $r : S \times A \rightarrow R$ is a reward function defined for any state $s \in S$ and any action $a \in A$, $P(st+1|st, at)$ is a transition probability distribution, t is a discrete time step, and the scalar $0 < \gamma \leq 1$ is a discount factor. In contrast, to the standard MDP formulation, we also assume access to a failure flag f that is returned on termination of the MDP, i.e. task completion. The failure flag indicates whether the desired task was completed successfully or not, i.e. $f = 0$ indicates success, $f = 1$ otherwise. For example, if the task is for a robot arm to reach to a target, then at termination $f = 0$ means the target was acquired, and $f = 1$ implies that the robot finished in a configuration far from the target. Thus, our modified MDP is denoted by $\langle S, A, r, f,P, \gamma\rangle$. In single-task RL, an AI agent generally learns a policy $\mu(a|o)$ choosing an action a given an observation o (e.g. from an image) of the state s. The agent’s objective is to determine the sequence of actions that maximizes the expected return E [P t γ t r(st, at)]. The specific task and its associated rewards are determined by the reward function r. In our case, we assume access to an atomic action library that corresponds to having a set of N pre-trained or pre-defined policies {µbi} N i=1 ready to use, each based on an underlying modified MDP. Note, we use hat µb to denote that the policy is trained/defined. Humans exhibit remarkable proficiency in synthesizing complex behaviors by composing various known skills. With regard to a robotic system, this process involves the composition of distinct policies that are executed following a certain structure, such as some appropriate sequential order, or corresponding to a data structure like a behavior tree. In situations when there is access to a team of experts, it is reasonable to assume that they are capable to define some reward function that measures appropriate compositions of atomic actions, i.e. behaviors. With access to experts, we can reasonably assume some reward or fitness function that specifies an appropriate composition of these policies, in which case we could explore methods based on hierarchical reinforcement learning [1]. However, in our case, absent expert guidance, the robotic system must rely on environmental observations o and non-expert human input h to guide policy selection. Also, we assume the input from the human is given by text.

3.3 Human non-expert interface We provide a chat-based interface to our framework to allow easy adoption from non-expert human users. Each environment step is executed after the human feedback is received from the interface, and then once the execution is over on the system (ending either with a success or a failure), we ask the human to input a new textual entry. At the beginning, we let the system interpret the first human input as the task description. The task description should outline the goal or objective to assign to the robot, providing context for the subsequent actions to generate. Thereafter, the system treats human input prompts as a feedback, which may contain suggestions for corrective behavior or suggestions for alternative approaches for the robot to complete the task. Another potential interaction mode could be via speech, which would have the potential to be even more intuitive for non-experts. We actually plan to implement a microphone into our setup and use an off-the-shelf audio-to-text package for parsing the input. This functionality will be incorporated into our main code-base in the future.

3.4 Prompt generation The prompt provided to the language model serves as input to generate behavior representations that can be executed on the system. At each environment step τ , the prompt is updated, ensuring that the language model receives the latest information necessary for decision-making. We show in the central part of Figure 2 the different elements that we expect in a prompt to shape the behavior of the system. The prompt includes a task description that is provided by the user, as described in the previous sub-section. After the first environment step τ , the non-expert provides feedback that the system uses to correct its behavior. A description of the atomic action library is also included to provide context on the admissible behaviors of the system, as described in Section 3.1. Moreover, the prompt contains an observation of the environment that is collected by mapping several sensor readings to text. Several well-known prompt engineering strategies are utilized to aid the language model construct a behavior for the system, namely chain-of-thought and few-shot prompting. These additional portions of the prompt are assumed to be given as part of all observations o. Finally, some additional notes are written in the prompt, such as how the language model should format the behavior output (e.g. Python or XML). Overall, the prompt generation process gathers information from both the ROS environment and the human interface, ensuring that the language model receives comprehensive input to guide its decision-making process. Once the prompt is constructed, it is then passed to the language model. We consider the output of the language model to represent the desired behavior for the system. We describe next the different formatting options for the output of the language model. 3.5 Behavior representation We call a behavior the combination of atomic actions that is extracted from the textual output of the language model. To represent a behavior, the LLM generates either a Python or XML code. When Python format is used, a Python terminal exposed to the ROS environment executes the LLM output. In the case of XML, the LLM response is interpreted as a behavior tree1 . We use regular expressions to easily identify parse LLM output that should encapsulate the code into ‘‘‘python...’’’, ‘‘‘json...’’’, or ‘‘‘xml...’’’. Python output We expect the use of Python code when the action library is a set of Python function that exposes the various functionality of the system. The library can also contain ROS actions and ROS services that can be interfaced with the script. JSON output When the JSON format is used, a behavior representation called an action sequence is used. In this case, the specified actions in the sequence are executed one after the other, and we expect each action to be a ROS service of the type rosllm_srvs/AtomicAction. This service returns a string called output and takes as input one string argument called input, which takes input per action, and another string argument prev_output, which is the output from the previous action. XML output A behavior tree, represented by XML code, is a hierarchical model that describes the behavior of autonomous agents or robots. It consists of nodes that define specific actions or conditions, and it is organized in a tree-like structure. At the root of the tree, the behavior selector node determines the order in which to evaluate and execute child nodes. These child nodes can include sequences of nodes, which execute their child nodes sequentially until one fails, or parallel nodes, which execute their child nodes simultaneously. Other types of nodes include conditional nodes, action nodes, and decorator nodes, each serving distinct roles in controlling the agent’s behavior. In this case, the language model is tasked with producing the XML code defining a behavior tree.

the capability of our framework to adapt and recover from such changes through continual learning facilitated by human feedback. The primary goal is to showcase the system’s ability to utilize human feedback to recover from unforeseen disruptions in task execution and subsequently learn from these experiences to handle similar changes in the future autonomously. The experimental protocol is as follows. 1. Task Specification: The system is tasked with “pick and place the box,” a common robotic manipulation task. 2. Action Sequence Generation: Using the provided task description, the language model generates an action sequence for execution. 3. Execution and Disruption: As the robot executes the task, the environment is intentionally perturbed by moving the target box, leading to a failure to grasp the object. 4. Human Feedback: Upon observing the failure, the human provides corrective feedback, advising the system to ensure the box’s proximity before grasping and then to retry the task. 5. Recovery and Adaptation: Leveraging the feedback, the system adapts its approach and successfully completes the task, demonstrating its ability to recover from environmental changes through human-guided learning. To evaluate the system’s capability for continual learning, a second trial is conducted under identical conditions, with the box moved simultaneously as in the previous trial. However, this time, the human feedback from the initial trial is incorporated into the task prompt. By doing so, we aim to assess whether the system autonomously applies the learned corrective action to handle similar environmental changes without human intervention

8 Conclusions In this section, we provide an overview of our main conclusions, acknowledge the current limitations of our framework and highlight potential future directions.

<u>8 - ROS-LLM: A ROS framework for embodied AI with task feedback and structured reasoning</u>

The human ability to learn, generalize, and control complex manipulation tasks through multimodality feedback suggests a unique capability, which we refer to as dexterity intelligence. Understanding and assessing this intelligence is a complex task. Amidst the swift progress and extensive proliferation of large language models (LLMs), their applications in the field of robotics have garnered increasing attention. LLMs possess the ability to process and generate natural language, facilitating efficient interaction and collaboration with robots. Researchers and engineers in the field of robotics have recognized the immense potential of LLMs in enhancing robot intelligence, human-robot interaction, and autonomy. Therefore, this comprehensive review aims to summarize the applications of LLMs in robotics, delving into their impact and contributions to key areas such as robot control, perception, decision-making, and path planning. We first provide an overview of the background and development of LLMs for robotics, followed by a description of the benefits of LLMs for robotics and recent advancements in robotics models based on LLMs. We then delve into the various techniques used in the model, including those employed in perception, decision-making, control, and interaction. Finally, we explore the applications of LLMs in robotics and some potential challenges they may face in the near future. Embodied intelligence is the future of intelligent science, and LLMs-based robotics is one of the promising but challenging paths to achieve this.

In conclusion, the applications of large language models in robotics hold tremendous potential. They provide new paradigms and methods for robot control, path planning, and intelligence. Through more intuitive and natural human-machine interaction, language-based path planning, and intelligent semantic understanding, large language models not only enhance the performance and efficiency of robots but also improve the experience and interaction modes of human-robot interaction. Therefore, this comprehensive review aims to summarize the applications of LLMs in robotics, delving into their impact and contributions to key areas such as robot control, perception, decision-making, and path planning. To summarize, there are four key contributions in this paper, as follows: • We discussed the latest advancements in LLMs and their significant impact on the field of robotics. We highlighted the benefits of LLMs for robots, as well as the emergence of new robot models equipped with LLMs in recent years. • We discussed the current state of robot technology, focusing on advancements in perception, decisionmaking, control, and interaction combined with LLMs. Specifically, we highlighted the critical role of LLMs in decision-making modules, which have enabled robots to make more informed and effective decisions in various applications. • We explored potential applications of current robots equipped with LLMs in the near future. • We discussed several potential challenges that robots may face when integrated with LLMs, as well as the potential impact of future developments in this field on human society.

affordance function. It will perform the most likely action to succeed in the current environment and state. For instance, upon receiving the instruction, “Can you help me get an apple?". LLM may decompose it into several tasks: “walking to the kitchen, opening the refrigerator, obtaining the apple, and delivering it to the requester.", just like in Figure 2(a). 2.3.2. PaLM-E While LLMs have demonstrated remarkable capabilities in handling complex tasks, integrating them as an interface into robots remains a significant challenge. A major limitation of LLMs is their reliance on text input, which is insufficient for robots that require physical interaction. PaLME [34] boasts an LLM capable of integrating continuous sensory information from the real world, effectively bridging the gap between language and perception. Its multi-modal input encompasses vision, text, and state estimation, like in Figure 2(b), as exemplified by the question "What is it in ?" The model’s processing is end-to-end, whose performance is state-of-the-art in OK-VQA [84]. PaLM-E is a visual-language generalist. PaLM-E treats images and text as multi-modal inputs represented by latent vectors. PaLME is a decoder-only model that generates text completions autonomously when provided with a prefix or hint. The output of PaLM-E is separated into two parts: when tackling text generation tasks (such as embedded question answering or scene description), the model directly produces the final output (i.e., output text or speech). In contrast, when utilized for specific planning and control tasks, PaLM-E generates low-level instruction text (e.g., instructions for controlling robot meta-actions). 2.3.3. LM-Nav Goal-based robot navigation can leverage large, unlabeled datasets for training, resulting in strong generalization capabilities in real-world scenarios. However, in visionbased settings, specifying targets often requires images. Current supervised learning methods are not only expensive but also demand linguistically described and labeled trajectory data, making them impractical for widespread use. How can users communicate with robots more conveniently? To address the challenge, LM-Nav [117] was developed, exploiting the advantages of language to facilitate effective communication between users and robots. The LM-Nav system comprises three components: a vision-navigation model (VNM); a visual-language model (VLM); and a large-scale language model (LLM). Notably, LM-Nav operates without the requirement of labeled data or fine-tuning. By leveraging the VLM and VNM, LM-Nav can extract landmark names from commands and navigate to specified locations. LMNav leverages three pre-trained models to achieve successful navigation in pre-explored environments. First, it employs ViNG [114] as a VNM creates a topological map using observations from a prior exploration of the environment. Subsequently, GPT-3 [27] serves as the LLM [14] processes free-form text instructions to determine the target landmark. Finally, CLIP [99] serves as the VLM to locate the corresponding position in the topology map based on the identified landmark. By combining these models, LMNav can effectively follow natural language instructions to complete navigation tasks. 2.3.4. Expedition A1 Expedition A12 , developed by AGIBot, embodies the company's commitment to seamlessly integrating advanced AI into robotics and fostering harmonious collaboration between humans and machines. Envisioning a future where robots serve as indispensable assistants to humans, AGIBot's mission is to create intelligent and versatile robots

3.3. Control Here, we argue that the control module is the key component responsible for regulating robotic actions. This module plays a crucial role in ensuring that the robot's actions are executed accurately and successfully, with a focus on the hardware aspects of action execution. 3.3.1. How to learn language-conditioned behavior Much of the previous work has focused on enabling robots and other agents to comprehend and execute natural language instructions [19, 35, 81]. There are various approaches to learning linguistically conditioned behaviors, such as image-based behavioral cloning that follows the BCZ [58] method or the MT-Opt [64] reinforcement learning method. Imitation learning techniques train protocols on demonstration datasets [58, 153], while offline reinforcement learning has also been studied extensively [59, 71, 88]. However, some works suggest that imitation learning on demonstration data performs better than offline reinforcement learning [83], and other studies demonstrate the feasibility of offline reinforcement learning in theory and practice [72, 73]. Many works combine RL and Transformer structures [20, 60], and some works integrate imitation learning with reward conditions, such as Decision Transformer (DT) [20], namely combines imitation learning with reinforcement learning elements. However, DT does not enable the model to learn from the demonstration dataset to have better performance. Deep Skill Graphs (DSG) [5] present a novel approach to skill learning utilizing the option framework. This method leverages graphs to represent discrete aspects of the environment, enabling the model to acquire structured knowledge and learn complex skills within the given domain. CT employs goal-conditioned RL to transform the local skill-learning problem into a goal-conditioned Markov decision process (MDP) [61]. In the context of navigation robots, early approaches to enhancing navigation strategies with the natural language employed static machine translation [80] to discover patterns. The process involves utilizing discovery patterns to translate free-form instructions into formal languages that adhere to specific grammatical rules [19, 85, 127]. However, these methods were limited to structured state spaces. Recent works have also developed the VLN task as a sequence prediction problem [3, 87, 119]. Additionally, there are methods that leverage nearly 1M labeled simulation trajectory demonstration data for training [47], but applying these models in unstructured environments remains a significant challenge. Data-driven approaches for vision-based mobile robot navigation often depend on the utilization of realistic simulation techniques [70, 111, 144] or gathering supervised data to directly learn policies for achieving goals based on observations [38]. Alternatively, self-supervised learning methods can utilize unlabeled datasets or trajectories generated automatically by onboard sensors and hindsight relabeling learning [51, 63, 114]. 3.3.2. How to execute action after parsing nature language To determine whether a skill can be executed in the current state after parsing a natural language command, a temporal-difference-based (TD) reinforcement learning approach can be employed. This method learns a value function to evaluate whether the skill is executable or not [1]. The value function is derived from the corresponding affordance function of reinforcement learning [42]. Additionally, LM-Nav [117] utilizes a self-supervised learning method to enhance the parsing of free-form language instructions leveraging pre-trained VLM in a large number of previous environments. To address the challenges of long-term tasks, hierarchical reinforcement learning (HRL) [55] can be employed, where higher-level policies play a role in setting objectives for lower-level protocols to execute [90, 132]. The process of mapping natural language and observations into robot actions can also be viewed as a sequence modeling problem [9, 10, 29]. Transformer-based robot control, such as the Behavior Transformer [113], focuses on learning demonstrations that correspond to each task. Gato [104] suggests training a model on large datasets including robotic and non-robotic. 3.4. Interaction Interaction serves as a fundamental module that enables robots to engage and interact with both the environment and humans. To enhance robots' ability to interact in the physical world, they are often trained extensively. While some researchers utilize artificial intelligence to interact in virtual environments, such as games or simulations, ultimately, these models must be transferred to the real world. F. Zeng et al.: Preprint submitted to Elsevier Page 10 of 19 Large Language Models for Robotics: A Survey However, the accuracy of these models tends to be lower in real-world settings compared to simulated environments.

In this survey, we summarized the methods and technologies currently used for large models in robots. First, we reviewed some basic concepts of large language models and common large models. We explain what improvements will be brought to robots by using large models as brains. We also introduce the representative LLM-based robot models proposed in recent years, such as LM-Nav [117], PaLMSayCan [1], PaLM-E [34], etc. Next, we divide the robot into four modules: perception, decision-making, control, and interaction. For each module, we discuss the relevant technologies and their functions, including the perception module's ability to process the robot's input from the surroundings; the decision-making module's capacity to understand human instructions and plan; the control module's role in processing output actions; the interaction module's ability to interact with the environment. We also explore the potential application scenarios of current robots based on LLMs and discuss the challenges, such as training, safety, shape, deployment, and long-term task performance. Finally, we consider the social and ethical implications of post-intelligent robots and their potential impact on human society. As LLMs continue to evolve, robots may become increasingly intelligent and capable of processing instructions and tasks more efficiently. With advancements in hardware, robots may eventually become reliable assistants for humans, as depicted in science fiction movies. However, we must also be mindful of their potential impact on society and address any concerns proactively. Embodied intelligence is a new paradigm for the development of intelligent science and is of great significance in leading the development of the future. LLM-based robotics represent a potential path to embodied intelligence. We hope this survey can provide some inspiration to the community and facilitate research in related fields.