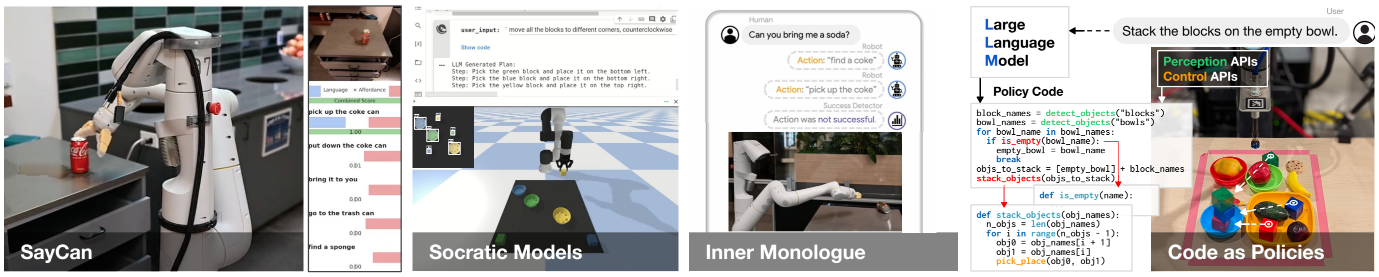
Robotics: Science and Systems 2023
Daegu, Republic of Korea, July 10-July 14, 2023

Demonstrating Large Language Models on Robots

Google DeepMind



```
user_input: "move all the blocks to different corners, counterclockwise
```

```
LLM Generated Plan:
Step: Pick the green block and place it on the bottom left.
Step: Pick the blue block and place it on the bottom right.
Step: Pick the yellow block and place it on the top right.
```

```
block_names = detect_objects("blocks")
bowl_names = detect_objects("bowls")
for bowl_name in bowl_names:
  if is_empty(bowl_name):
    empty_bowl = bowl_name
    break
objs_to_stack = [empty_bowl] + block_names
stack_objects(objs_to_stack)
```

```
def is_empty(name):
  ...
```

```
def stack_objects(obj_names):
  n_objs = len(obj_names)
  for i in range(n_objs - 1):
    obj0 = obj_names[i + 1]
    obj1 = obj_names[i]
    pick_place(obj0, obj1)
```

Fig. 1: Our proposed demonstration presents a collection of systems that use large language models on robots, with four complementary approaches. This includes an accessible video chat interface that allows users to (i) type in natural language instructions to different robots connected remotely and view them as they execute the task, and (ii) seamlessly switch between systems and hardware platforms. Our goal is to communicate their limitations, discuss open research problems in the area, and provide tooling (open-source code) to empower other researchers to explore exciting new opportunities in the space.

***Abstract*—Robots may benefit from large language models (LLMs), which have demonstrated strong reasoning capabilities across various domains. This demonstration includes several systems based on recent methods that integrate LLMs on robots: SayCan [1], Socratic Models [49], Inner Monologue [19], and Code as Policies [27]. While each algorithm highlights a different mode of grounding, they all share a common system-level structure in that they use LLMs to take as input natural language instructions and generate robot plans in the form of step-by-step procedures or code. This structure provides several practical perks for demonstration in that (i) we can use existing video chat interfaces to instruct the robot by typing commands and broadcasting its movements in action via video streaming, (ii) one can seamlessly switch between systems by switching between interfaces that communicate with different robots, and (iii) this can all be done remotely on a laptop, where the robots on real hardware can be held on standby in the lab ready to run on command. Our tentative plan is to show at least one system running on real hardware remotely – Inner Monologue [19] or Code as Policies [27], and solicit task instructions from the live audience. Time-permitting we may also demonstrate the other systems available to run on real hardware. Otherwise, we will present recorded videos of past runs. We will link to open-source code, and conclude with a discussion of open research questions in the area.**

I. Introduction

Robots may benefit from large language models (LLMs) [9, 7], which have demonstrated strong reasoning capabilities across various domains, including strategic dialogue [4], step-by-step reasoning [46, 24], math problem solving [26, 35], and code writing [8]. While LLMs possess vast amounts of knowledge acquired from pre-training on Internet-scale text data, they lack grounding (situated) in the physical world [43]. Enabling these models to bridge the connection between words, percepts, and actions on robots remains an open research question.

In this work, we propose a demonstration of several systems based on recent methods that integrate LLMs on robots: SayCan [1], Socratic Models [49], Inner Monologue [19], and Code as Policies [27] – providing an in-depth discussion on how they differ and build on each other. While each algorithm highlights a different mode of grounding e.g., using affordance functions or visual language model (VLM) outputs, they all share a common system-level structure in that they use LLMs to take as input natural language instructions and generate robot plans in the form of step-by-step procedures or code. This structure provides several practical perks for demonstration in that (i) we can use existing video chat interfaces to instruct the robot by typing commands and broadcasting its movements in action via video streaming, (ii) one can seamlessly switch between systems by switching between interfaces that communicate with different robots, and (iii) this can all be done remotely on a laptop, where the robots on real hardware can be held on standby in the lab ready to run on command.

Our tentative plan is to show at least one system running on real hardware remotely – Inner Monologue [19] or Code as Policies [27], and solicit task instructions from the live audience. For the other systems, we will present recorded videos of past runs, and link to open-source code (in simulation). Time-permitting (depending on the structure of the session), we may also demonstrate the other systems available to run on real hardware. These can include: (i) examples of common failure modes that occur during these demonstrations (Sec. IV), and (ii) "experimental" demonstrations (Sec. V) of using LLMs for robot reasoning that were not robust enough (either on physical robots, or across multiple LLMs) to appear in a publication, but that may lead to new capabilities as LLMs continue to evolve and improve.

Our primary goal is to communicate to the research community that while integrating LLMs on robots can lead to interesting new capabilities, there is still much more work to be done in the area. Asking the audience for input task instructions is one way we'd like to probe our existing systems live and discuss failure modes as they naturally arise. We'd like to empower the audience with the tools that they need to execute on the research. To this end, every system will come with available open-source code for researchers to extend and build on, with colab notebooks (including simulation) that can be directly run in the browser. We conclude with several open research questions in the area.

II. Related Work

Controlling robots with language has a long history, including early demonstrations of human-robot interaction through lexical parsing of natural language [47]. Language serves not only as an interface for non-experts to interact with robots [5, 25], but also as a means to compositionally scale generalization to new tasks [21, 1]. The literature is vast (we refer to Tellex et al. [43] and Luketina et al. [28] for comprehensive surveys), but recent works fall broadly into the categories of high-level interpretation (e.g., semantic parsing [30, 25, 44, 42, 38, 31, 45]), planning [18, 19, 1], and low-level policies (e.g., model-based [33, 3, 39], imitation learning [21, 29, 40, 41], or reinforcement learning [22, 16, 10, 32, 2]). In contrast, our work focuses on using existing pretrained LLMs to generated plans, procedures, and code as an expressive way to control the robot.

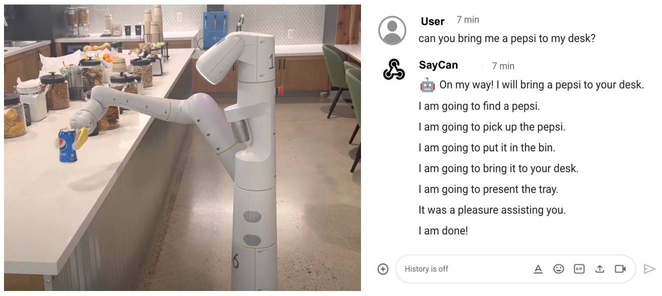
Large language models exhibit impressive zero-shot reasoning capabilities: from planning [18] to writing math programs [13]; from solving science problems [26] to using trained verifiers [11] for math word problems. These can be improved with prompting methods such as Least-to-Most [50], Think-Step-by-Step [24] or Chain-of-Thought [46].

III. Proposed Demonstrations

Overview. In this section, we describe a collection of systems – each using LLMs within the algorithm to enable new capabilities and a natural language interface. We describe how the LLMs are used, and how each system differs from and builds on each other. Each demonstration comes with existing videos and open-sourced code that will be presented. Since each corresponding system can be instructed to perform open-query tasks given arbitrary natural language instructions, this provides an opportunity to easily present demonstrations of multiple systems under a common user interface that can be thought of as "video chat with robots".

Format. The extent to which we present each demonstration would be conditioned on the length of time available to us during the conference session. For example, if allotted 5-10 minutes of presentation time, then we plan to present a live demonstration of Inner Monologue (Sec. III-C) or Code as Policies (Sec. III-D), with an Everyday Robot running remotely in our lab during the presentation. The presenter would provide a brief high-level overview of the project(s), then switch during the presentation to a live video feed where both the robot and a chat dialogue interface are visible. They would type natural language instructions in the chat (that are then communicated via gRPC) to the robot to perform the task. For example, one such chat-based interaction may look like the following:

The remainder of the presentation time would be spent on (i) (time permitting) opening up to live instructions and queries suggested by the audience, and (ii) referencing website and download links to videos and open-source code available for the other projects. Alternatively, if we are instead allotted a booth (in parallel to conference sessions) or a prolonged poster session (30min+), then we plan to divide the time among all demonstration systems that we have available (described in the following sections), and/or present specific systems on request from the audience. Our robot systems (and operations team) for SayCan (Sec. III-A), Inner Monologue (Sec. III-C), and Code as Policies (Sec. III-D) would be remote on standby in our office locations, ready to run on demand.



User 7 min
can you bring me a pepsi to my desk?

SayCan 7 min
🤖 On my way! I will bring a pepsi to your desk.
I am going to find a pepsi.
I am going to pick up the pepsi.
I am going to put it in the bin.
I am going to bring it to your desk.
I am going to present the tray.
It was a pleasure assisting you.
I am done!

Fig. 3: SayCan is accessible via a chat-based interface which engages in a dialog with a user to communicate it's abilities and determine the task, then executes the instruction in the real world on a mobile manipulator.

Simulated environments Inner Monologue (Sec. III-B) would be demonstrated live in the browser. Onsite at the conference, only a laptop should be needed to present an interactive demonstration (as a video chat with the robot). By soliciting queries from participating audience members, we hope to openly communicate the performance and limits of the algorithms, and use their failure modes as a platform to discuss open areas of research. Specifically, our demonstration includes running examples of language-using robots using various LLMs (PaLM [9], GPT-3 [7, 34], Codex [8]) with VLMs (CLIP [36], ViLD [17], MDETR [23]) on two platforms (EverydayRobots and UR5) in both simulated and real settings.

A. *SayCan: LLM Planning with Affordance Functions*

Given natural language instructions as input (in green), SayCan uses LLM planning [18] and grounding affordance functions to generate a step-by-step sequential plan (highlighted):

```
Instruction: Move the coke can a bit to the right.
Step 1. Pick up coke can.
Step 2. Move right.
Step 3. Place coke can.
```

Each step in the plan is selected iteratively based on a combination of a their corresponding probability scores from a language model (that judges the utility of the task), and the affordance values of a grounding model (that judges how feasible a task is) conditioned on input images. The plan is then executed by pretrained, natural-language conditioned policies that can either be trained with RL, or with supervised learning, though for this demonstration they are implemented in the real world via RT-1 [6] and in a simulator via CLIPort [40].

The demonstration consists of a real world mobile manipulator from Everyday Robots, operating in an office kitchen stocked with various objects. The demo proceeds through a chat interface with the language model, which at first will ask for the task, then clarify any ambiguity and inform you if the task is not feasible. Once the task is agreed upon, it will execute it with SayCan for high-level planning and RT-1 for low-level skills (with a live video of the robot). At each step, it will output the affordance and language model scores for the top skill opens via an interpretable interface. More information and code can be found at the project website say-can.github.io.

B. *Socratic Models: VLM-informed LLM Planning*

Rather than post-hoc grounding generated steps to skills, Socratic Models uses pretrained visual language models (VLMs) (e.g., open-vocabulary object detectors [23, 17]) to provide additional

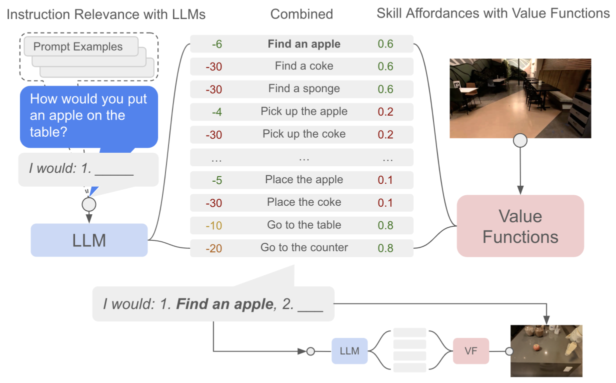


Fig. 4: Given a high-level instruction, SayCan combines probabilities from a LLM (the probability that a skill is useful for the instruction) with the probabilities from a value function (the probability of successfully executing said skill) to select the skill to perform. This emits a skill that is both possible and useful. The process is repeated by appending the skill to the response and querying the models again, until the output step is to terminate.

context (e.g., text descriptions of images) to the LLM before generating the step-by-step plan. Each step is formatted as a function call to a pre-existing skill (parameterized with text) that calls a language-conditioned policy such as CLIPort [40]:

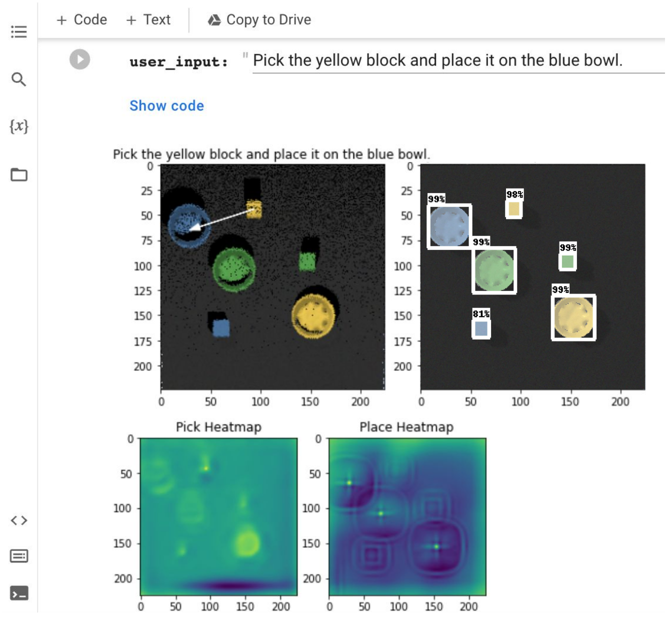
```
# Move the coke can a bit to the right.
objects = ["coke can"] from open-vocab VLMs
Step 1. robot.grasp("coke can")
Step 2. robot.place_a_bit_right()
```

One interpretation of Socratic Models is that it uses text as the information bottleneck between modules (perception and planning), as opposed to a Bayesian approach to multimodal probabilistic inference. This presents both advantages and disadvantages – while the expression of supported tasks can expand to a broad spectrum of objects (detectable by the VLM) and skills (executable by the language-conditioned policies) via compositional generality [20], language may struggle to succinctly describe contact-rich interactions or dynamics (particularly those not supported by existing VLMs or language-conditioned policies).

The demonstration of Socratic Models simulates a UR5 robot arm overlooking a set of objects (blocks and bowls) on a tabletop setting in PyBullet [12]. It uses ViLD [17] to detect the objects in the scene, then passes that information as a list of object names to InstructGPT [34] as the LLM planner, which is few-shot prompted to generate function calls to CLIPort. The function calls themselves map to a text template that matches the input training data distribution of CLIPort. The demonstration can be presented directly within a colab that runs in browser, where intermediate outputs can be visualized for prototyping. More information and code can be found on the project website socraticmodels.github.io.

C. *Inner Monologue: LLM Planning with Feedback*

SayCan and Socratic Models both do not use feedback to re-plan. Inner Monologue addresses this by expanding LLM step-by-step planning (SayCan) to include language-based feedback (Socratic Models), in the form of success detection or scene description, which can come from pretrained VLMs or human feedback:



user_input: " Pick the yellow block and place it on the blue bowl.

Fig. 5: Socratic models proposes a modular framework to enable multimodal foundation models (vision, language, robotic, and more) to exchange information and flexibly execute downstream capabilities. Above, an LLM planner [1, 18], CLIP-based object detector [17], and CLIPort policy [40].

```
Instruction: Move the drink a bit to the right.
Objects: water bottle, coke can. from open-vocab VLMs
Do you want to move the water bottle or the coke can?
Human: Coke can please.
Step 1. Pick up coke can.
Success detector: action was not successful.
Step 1. Pick up coke can.
Success detector: action was successful.
Step 2. Move right.
Step 3. Place coke can.
```

The demonstration consists of a real world mobile manipulator from Everyday Robots, operating in an office kitchen stocked with various objects. The demo receives a task via a chat interface and begins executing it with SayCan. A robot operator overseeing the demonstration provide both a video link and interact with the robot, forcing skill failures randomly and changing the environment to demonstrate the ability of Inner Monologue to react to unforeseen changes. At each step, the chat interface will output the results of the success detector or scene description and enable the user to continue interacting with the robot, potentially changing the task during execution. More information can be found at the project website innermonologue.github.io.

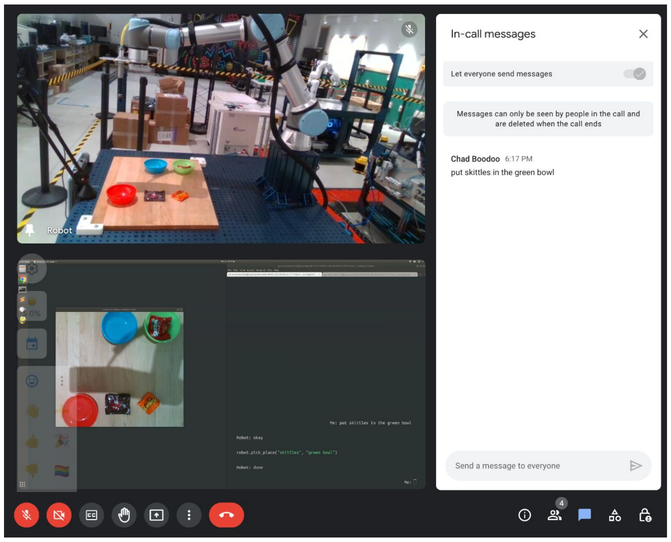
Instruction VLM-informed Data Augmentation Learning-based robotic control systems rely on large datasets of diverse behavior trajectories, such as those collected with teleoperation. VLMs and LLMs can be utilized for language instruction prediction, enabling a cheap and scalable approach for importing internet-scale semantic knowledge into offline robotic datasets. More information can be found at the project website instructionaugmentation.github.io.

D. *Code as Policies: LLM Planning with Code*

While LLMs can generate plans in the form of a list of sequential steps, Code as Policies uses code-writing LLMs to expand this to the full expression of code: including if-else conditionals, for/while loops, arithmetic, third-party libraries, etc. This can encompass both re-planning with feedback loops, low-level and spatial reasoning:

```
# Move the coke can a bit to the right.
while not obj_in_gripper("coke can"):
   robot.move_gripper_to("coke can")
robot.close_gripper()
pos = robot.gripper.position
robot.move_gripper(pos.x, pos.y+0.1, pos.z)
robot.open_gripper()
```

Code as Policies assumes that a collection of perception and control primitive APIs are available as modules, and that the code-writing LLM has been exposed to several examples (via few-shot prompting) of the function calls in use, so that it can re-compose them for new tasks. The generated code is directly executed on the robot.



Messages can only be seen by people in the call and are deleted when the call ends

Chad Boodoo 6:17 PM
put skittles in the green bowl

```
Me: put skittles in the green bowl
Robot: okay
robot.pick_place("skittles", "green bowl")
Robot: done
Me:
```

Fig. 6: The chat interface for Code as Policies enables a user to request new tasks and view both the robot execution, robot view, and the code generated.

The demonstration consists of a real UR5 robot arm overlooking a tabletop workspace with a set of objects, with a chat UI that includes a video feed of the robot workspace, and displays generated code by the LLM. More information, open-source code (in simulation), and a hugging face demonstration can be found at the project website code-as-policies.github.io.

IV. Common Failure Modes

Robot systems built with LLMs as a central component are not without practical limitations. Here, we discuss several chronic issues (both methods-level and systems-level) that may arise during demonstrations, which may lead to new infrastructure considerations for future work.

- LLMs are compute-heavy e.g., GPT-3 [7] is a 175 billion parameter model, while PaLM [9] is a 540 billion parameter model. Both require large remote GPU and TPU server farms to run inference alone. Prediction services in the cloud are subject to latencies due to Internet bandwidth, or concurrent usage, and may disconnect in the middle of a demonstration.
- The scope of generalization provided by LLM planning (with the aforementioned methods) are also subject to the hand-designed few-shot prompt examples provided in-context as input to the LLM. These examples must often encompass the "convex hull" of tasks instructions that will be asked at test time, which can be challenged by hard tasks requested by a live audience. We hope to use demonstrations and live queries as a means to surface these issues.
- LLMs are subject to biases from their training data. This can lead to diverging behaviors between different LLMs when tasked with the same ambiguous/subjective instructions (e.g., "put the block in the bowl you think it best fits"). This can also lead to practical implications on the expression of the LLM plans. For example, Codex [8] may perform well with generating Python code or pseudo-code (given that it was trained on Python Github code), but may struggle to be as expressive in other languages (e.g., C++) or formats (e.g., temporal logic) without addition model finetuning.
- We inherit other limitations of LLMs. One typical issue is that LLMs use auto-regressive decoding, which means previous errors will act as a input for future output. This could happen in a dialog setting and makes an LLM agent enters an error state (e.g. denying all requests by saying "I cannot do that."). This could happen in a live demo and usually flushing the dialog history and restart the demo would be helpful.

V. Experimental Demonstrations

This section describes additional interesting examples of LLMs for robot reasoning that can be demonstrated on request, but are not robust enough (either on physical robots, or across multiple LLMs) to appear in a main project. Nevertheless, these examples may serve as a point of discussion for seeking potential new capabilities as LLMs continue to evolve and improve.

A. *Cross-Embodied Code as Policies*

Code-writing LLMs exhibit a degree of cross-embodiment support [48, 14] by performing the same task differently depending on the action APIs. In the example below, we provide brief docstrings of the action APIs (in comments), and the resultant plan changes depending on the whether or not the robot is omnidirectional or unidirectional. We note that this ability is brittle with existing LLMs and cannot reliably adapt to APIs that are very different. More robustness may require larger ones trained on domain-specific code.

```
# omnidirectional robot.
# available actions: move_up(dist), move_right(dist),
#                    move_back(dist)
# make a U shape 5 meters wide and 10 meters long.
robot.move_back(dist=10)
robot.move_right(dist=5)
robot.move_up(dist=10)
# unidirectional robot.
# available actions: turn_left(deg), move_forward(dist).
# make a U shape 3 meters wide and 8 meters long.
robot.move_forward(dist=8)
robot.turn_left(deg=90)
robot.move_forward(dist=3)
robot.turn_left(deg=90)
robot.move_forward(dist=8)
```

VI. Limitations and Open Research Questions

To be set free in human-centric environments, LLM-backed robots must show strong generalization performance in the "wild".

This depends on the robustness of the underlying in-context learning mechanisms which may be susceptible to distribution shifts between the LLM training data and inference time prompts, or between in-context examples and the query specifying the task [15]. Understanding the "generalization radius", robustness and sample complexity of in-context learning is an open research question.

LLMs can decompose long-horizon tasks up to a finite resolution beyond which low-level embodiment-specific action primitives take over. How much on-robot data needs to be collected to complement the reasoning capabilities of pretrained models depends on how this interface is designed. Furthermore, large models are hosted and served using cloud resources; hence, controllers onboard the robot need to be able to mitigate the latency for highly dynamic tasks.

The scope of embodied reasoning is broad and encompasses the ability of physical agents, such as language-using robots, to both (i) interpret and describe their surroundings, and (ii) plan interactions within the environment, taking into account physical constraints [37]. While the community has made considerable progress, reasoning about these constraints in the most expressive way possible still remains a challenge.

ACKNOWLEDGEMENTS

This manuscript and demonstration has been prepared by: Brian Ichter, Fei Xia, Ted Xiao, Vikas Sindhwani, and Andy Zeng.

REFERENCES

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

[2] Ahmed Akakzia, Cédric Colas, Pierre-Yves Oudeyer, Mohamed Chetouani, and Olivier Sigaud. Grounding language to autonomously-acquired skills via goal generation. *arXiv:2006.07185*, 2020.

[3] Jacob Andreas, Dan Klein, and Sergey Levine. Learning with latent language. *arXiv:1711.00482*, 2017.

[4] Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624): 1067–1074, 2022.

[5] Cynthia Breazeal, Kerstin Dautenhahn, and Takayuki Kanda. Social robotics. *Springer handbook of robotics*, 2016.

[6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[7] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

[8] 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. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[9] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[10] Geoffrey Cideron, Mathieu Seurin, Florian Strub, and Olivier Pietquin. Self-educated language agent with hindsight experience replay for instruction following. *DeepMind*, 2019.

[11] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *arXiv:2110.14168*, 2021.

[12] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. 2016.

[13] Iddo Drori, Sarah Zhang, Reece Shuttleworth, Leonard Tang, Albert Lu, Elizabeth Ke, Kevin Liu, Linda Chen, Sunny Tran, Newman Cheng, et al. A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level. *PNAS*, 2022.

[14] Aditya Ganapathi, Pete Florence, Jake Varley, Kaylee Burns, Ken Goldberg, and Andy Zeng. Implicit kinematic policies: Unifying joint and cartesian action spaces in end-to-end robot learning. *arXiv:2203.01983*, 2022.

[15] Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What can transformers learn in-context? a case study of simple function classes. *arXiv preprint arXiv:2208.01066*, 2022.

[16] Prasoon Goyal, Scott Niekum, and Raymond J Mooney. Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. *arXiv:2007.15543*, 2020.

[17] Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. Open-vocabulary object detection via vision and language knowledge distillation. *arXiv:2104.13921*, 2021.

[18] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022.

[19] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, et al. Inner monologue: Embodied reasoning through planning with language models. *arXiv preprint arXiv:2207.05608*, 2022.

[20] Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: How do neural networks generalise? *JAIR*, 2020.

[21] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[22] Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *NeurIPS*, 2019.

[23] Aishwarya Kamath, Mannat Singh, Yann LeCun, Gabriel Synnaeve, Ishan Misra, and Nicolas Carion. Mdetr-modulated detection for end-to-end multi-modal understanding. In *ICCV*, 2021.

[24] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners. *arXiv preprint arXiv:2205.11916*, 2022.

[25] Thomas Kollar, Stefanie Tellex, Deb Roy, and Nicholas Roy. Toward understanding natural language directions. In *HRI*, 2010.

[26] Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022.

[27] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022.

[28] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob N. Foerster, Jacob Andreas, Edward Grefenstette, S. Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. In *IJCAI*, 2019.

[29] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.

[30] Matt MacMahon, Brian Stankiewicz, and Benjamin Kuipers. Walk the talk: Connecting language, knowledge, and action in route instructions. *AAAI*, 2006.

[31] Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. Learning to parse natural language commands to a robot control system. In *Experimental robotics*, 2013.

[32] Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. *arXiv:1704.08795*, 2017.

[33] Suraj Nair, Eric Mitchell, Kevin Chen, Silvio Savarese, Chelsea Finn, et al. Learning language-conditioned robot behavior from offline data and crowd-sourced annotation. In *Conference on Robot Learning*, pages 1303–1315. PMLR, 2022.

[34] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[35] Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. *arXiv preprint arXiv:2202.01344*, 2022.

[36] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[37] Nicholas Roy, Ingmar Posner, Tim Barfoot, Philippe Beaudoin, Yoshua Bengio, Jeannette Bohg, Oliver Brock, Isabelle Depatie, Dieter Fox, Dan Koditschek, et al. From machine learning to robotics: challenges and opportunities for embodied intelligence. *arXiv preprint arXiv:2110.15245*, 2021.

[38] Dhruv Shah, Blazej Osinski, Brian Ichter, and Sergey Levine. Lm-nav: Robotic navigation with large pre-trained models of language, vision, and action. *arXiv:2207.04429*, 2022.

[39] Pratyusha Sharma, Balakumar Sundaralingam, Valts Blukis, Chris Paxton, Tucker Hermans, Antonio Torralba, Jacob Andreas, and Dieter Fox. Correcting robot plans with natural language feedback. *arXiv:2204.05186*, 2022.

[40] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. Cliport: What and where pathways for robotic manipulation. In *Conference on Robot Learning*, pages 894–906. PMLR, 2022.

[41] Simon Stepputtis, Joseph Campbell, Mariano Phielipp, Stefan Lee, Chitta Baral, and Heni Ben Amor. Language-conditioned imitation learning for robot manipulation tasks. *NeurIPS*, 2020.

[42] Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. Understanding natural language commands for robotic navigation and mobile manipulation. In *AAAI*, 2011.

[43] Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. Robots that use language. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:25–55, 2020.

[44] Jesse Thomason, Shiqi Zhang, Raymond J Mooney, and Peter Stone. Learning to interpret natural language commands through human-robot dialog. In *IJCAI*, 2015.

[45] Jesse Thomason, Aishwarya Padmakumar, Jivko Sinapov, Nick Walker, Yuqian Jiang, Harel Yedidsion, Justin Hart, Peter Stone, and Raymond Mooney. Jointly improving parsing and perception for natural language commands through human-robot dialog. *JAIR*, 2020.

[46] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

[47] Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. *MIT PROJECT MAC*, 1971.

[48] Kevin Zakka, Andy Zeng, Pete Florence, Jonathan Tompson, Jeannette Bohg, and Debidatta Dwibedi. Xirl: Cross-embodiment inverse reinforcement learning. In *CoRL*. PMLR, 2022.

[49] Andy Zeng, Adrian Wong, Stefan Welker, Krzysztof Choromanski, Federico Tombari, Aveek Purohit, Michael Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, et al. Socratic models: Composing zero-shot multimodal reasoning with language. *arXiv preprint arXiv:2204.00598*, 2022.

[50] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models. *arXiv:2205.10625*, 2022.