Multimodal Pretrained Models for Verifiable Sequential Decision-Making: Planning, Grounding, and Perception

Yunhao Yang
University of Texas at Austin
Austin, TX, United States
yunhaoyang234@utexas.edu

Cyrus Neary
University of Texas at Austin
Austin, TX, United States
cneary@utexas.edu

Ufuk Topcu
University of Texas at Austin
Austin, TX, United States
utopcu@utexas.edu

ABSTRACT

Recently developed pretrained models can encode rich world knowledge expressed in multiple modalities, such as text and images. However, the outputs of these models cannot be integrated into algorithms to solve sequential decision-making tasks. We develop an algorithm that utilizes the knowledge from pretrained models to construct and verify controllers for sequential decision-making tasks, and to ground these controllers to task environments through visual observations with formal guarantees. In particular, the algorithm queries a pretrained model with a user-provided, text-based task description and uses the model's output to construct an *automaton-based controller* that encodes the model's task-relevant knowledge. It allows formal verification of whether the knowledge encoded in the controller is consistent with other independently available knowledge, which may include abstract information on the environment or user-provided specifications. Next, the algorithm leverages the vision and language capabilities of pretrained models to link the observations from the task environment to the text-based control logic from the controller (e.g., actions and conditions that trigger the actions). We propose a mechanism to provide probabilistic guarantees on whether the controller satisfies the user-provided specifications under perceptual uncertainties. We demonstrate the algorithm's ability to construct, verify, and ground automaton-based controllers through a suite of real-world tasks, including daily life and robot manipulation tasks.

KEYWORDS

Multimodal Pretrained Model; Sequential Decision-Making; Automaton-Based Representation; Formal Methods; Verification; Perception

ACM Reference Format:
Yunhao Yang, Cyrus Neary, and Ufuk Topcu. 2024. Multimodal Pretrained Models for Verifiable Sequential Decision-Making: Planning, Grounding, and Perception. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

1 INTRODUCTION

While the rapidly emerging capabilities of multimodal pretrained models (also referred to as foundation models or base models) in question answering, code synthesis, and image generation offer new opportunities for autonomous systems, a gap exists between the text-based and image-based outputs of these models and algorithms for solving sequential decision-making tasks. Additional methods are required to integrate the outputs of these pretrained models into autonomous systems that can perceive and react to an environment in order to fulfill a task. Additionally, it is hard, if not impossible, to formally verify whether autonomous systems implementing such pretrained models satisfy user-provided specifications.

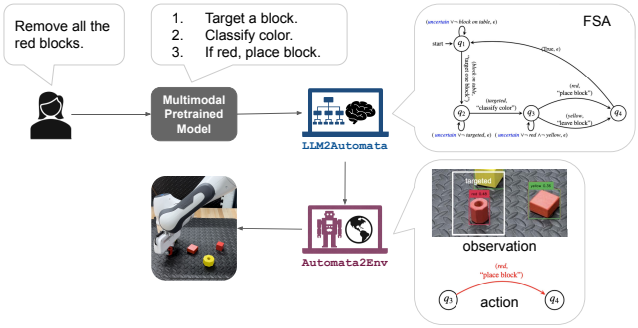


Figure 1: A real-world example that applies the proposed pipeline to a robot arm manipulation task.

Towards filling the gap between multimodal pretrained models and sequential decision-making, we develop a pipeline that integrates the outputs of pretrained models into downstream design steps, e.g., control policy synthesis or reinforcement learning, and provides a systematic way to ground the knowledge from such models. Specifically, we develop an algorithm to construct *automaton-based controllers* representing the knowledge from the pretrained models. Such representations can be formally verified against knowledge from other independently available sources, such as abstract information on the environment or user-provided specifications. This verification step ensures consistency between the knowledge encoded in the pretrained model and the knowledge from other independent sources. To implement the controllers in their task environments, we leverage the multimodal capabilities of the pretrained models, i.e., simultaneous vision and language understanding, to ground these controllers through visual perception.

The proposed method `Automata2Env` links image-based observations from the task environment to the controller's text-based propositions representing the environment's conditions. Specifically, `Automata2Env` collects visual observations and uses the vision and language capabilities of the employed pretrained models to evaluate the truth values of conditions from the controller. The controller then uses these truth values to select its next action. We

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

propose a mechanism that halts the controller's actions under perceptual uncertainties, i.e., potential misclassifications raised by the pretrained model. By doing so, this mechanism can provide probabilistic guarantees of whether the controller satisfies user-provided specifications while operating in the task environment. It helps to ensure the autonomous agent's safety with respect to provided mission specifications under perceptual uncertainties. For verification purposes, we use *finite state automata* (FSAs) to represent the controllers.

To construct these FSA-based controllers, we develop an algorithm named `LLM2Automata` that constructs a controller encoding the task knowledge obtained from the pretrained model. The algorithm builds upon the authors' recently presented algorithm, `GLM2FSA` [36]: It similarly queries the pretrained model to obtain text-based task knowledge, parses the text to extract actions, and defines a set of rules (grammar) to transform these actions into an FSA. In contrast to `GLM2FSA`, `LLM2Automata` explicitly queries the pretrained model for the environment conditions before and after each action is taken and encodes them into the constructed controller. This distinction of `LLM2Automata` is proposed to facilitate the grounding method `Automata2Env`, which connects these conditions to image-based observations of the task environment.

We demonstrate the algorithms' capabilities on sequential decision-making tasks through a variety of case studies. We provide proof-of-concept examples of commonsense tasks (e.g., cross the road) and real-robot tasks (e.g., robot arm manipulation). Figure 1 illustrates the major components of the proposed pipeline when it is applied to a robot arm manipulation task. These examples show the algorithms' ability to construct verifiable knowledge representations and to ground these representations in real-world environments through visual observations with perceptual uncertainties.

2 RELATED WORK

Formal Representations of Textual Knowledge. Many works have developed methods to construct symbolic representations of task knowledge from natural language descriptions. Several works construct knowledge graphs from textual descriptions of given tasks [10, 27, 35], or analyze causalities between the textual step descriptions and build causal graphs [20]. However, the graphs resulting from these works are not directly useful in algorithms for sequential decision-making, nor are they formally verifiable. Another work builds automaton-based representations of task-relevant knowledge from text-based descriptions of tasks [36]. These representations are both formally verifiable and directly applicable to algorithms for sequential decision-making. However, in contrast with [36], we not only generate automaton-based representations but also ground the generated representations to the task environment through image-based perceptions.

Multimodal Models in Sequential Decision-Making. A work [21] generates static high-level plans and matches them to the closest admissible action. Some other works [11, 17, 18, 30] generate zero-shot plans for sequential decision-making tasks from querying generative language models. These works require a set of pre-defined actions, which limit their generalization capability. Another work [33] uses large language models to generate executable code or API for robots. These works lack a procedure to ensure the correctness or safety of their generated plans or executable actions. In contrast, the automaton-based representation we constructed enables others to formally verify the plans against some mission or safety specifications.

Multimodal Models Grounding and Perceptions. Several works [12, 13, 29, 32] match textual plans to image observations and perform actions based on the perceptual outputs. A work [34] recursively generates plans based on visual observations. Other works [16, 22] match texts to images by generating visual-grounded textual plans from generative models and images.

However, none of these works guarantees the safety or correctness during the grounding procedure, especially under perceptual uncertainties. They assume the vision models can correctly classify the content within the image and correctly make plans or actions accordingly. In contrast, we consider the uncertainties in the image observations from the environments and enable the capability of formally verifying the automaton-based representations against provided specifications over the task environment with uncertainties.

There are multimodal pretrained models with vision and image capabilities that can interpret the content within the images and connect images to natural language. CLIP [24] measures the text-image consistency. Many other models [25, 26] can detect objects described in text from a given image. However, they have a fixed set of vocabularies to define objects. Open-vocabulary object detection models [8, 14, 15, 19] remove the constraints on vocabularies, which we will use for connecting the automaton-based representations to the task environment.

3 PRELIMINARIES

Multimodal Pretrained Models. Multimodal pretrained models (also referred to as foundation models [7] or base models [24]) are capable of processing, understanding, and generating data across multiple formats, such as images, text, and audio. These models are pretrained on large training datasets, and they have demonstrated strong empirical performance across a variety of tasks, such as question-answering and next-word prediction, even without further task-specific fine-tuning [2].

The Generative Pretrained Transformer (GPT) series of models [2, 23] consists of the most well-known multimodal pretrained models that can generate natural language or other data formats. In addition to GPT, pretrained models such as PaLM [5], BLOOM [28], Codex [4], and Megatron [31] also have the capability of generating outputs in natural language or other formats. Language generation is the core capability of these models, which we will use in the rest of the paper. Hence we denote this category of multimodal pretrained models as GLMs.

Vision-language models such as CLIP [24], Yolo [25], and the Segment Anything Model [14] are another type of multimodal pretrained model. CLIP takes an image and a set of texts as inputs, and measures the image-text consistency. Yolo, R-CNN [26] and Segment Anything Model are object detection models, which take an image and a set of words that describe objects, and classify whether the objects appear in the image. These models are capable of processing and understanding texts and images but are not capable of content generation.

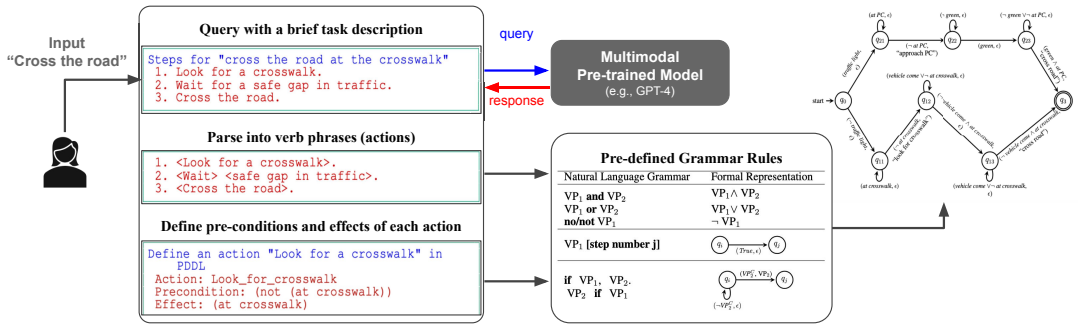


Figure 2: Demonstration of using the `LLM2Automata` algorithm to construct the controller through queries to the *large-scale generative language model* (GLM).

Finite State Automaton. A *finite state automaton* (FSA) is a tuple $\mathcal{A} = \langle \Sigma, \Gamma, Q, q_0, \delta, \omega \rangle$ where Σ is the input alphabet (the set of input symbols), Γ is the output alphabet (the set of output symbols), $q_0 \in Q$ is the initial state, $\delta : Q \times \Sigma \times Q \to \{0, 1\}$ is the transition function, and $\omega : Q \times \Sigma \times Q \to \Gamma$ is the output function.

We use P to denote the set of atomic propositions, which we use to define the input alphabet, $\Sigma := 2^P$. In words, any given input symbol $\sigma \in \Sigma$ consists of a set of atomic propositions from P that currently evaluate to True. A *propositional logic formula* is based on one or more atomic propositions in P. A transition from q_i to q_j exists if $\delta(q_i, \varphi, q_j) = 1$, the current state is q_i, and the propositional logic formula φ is true. Note that we define the FSA transitions to possibly be non-deterministic, i.e., multiple transitions are possible under the same input symbol from a given FSA state.

Controllers and Models. In this work, we refer to the automaton-based representation of task knowledge as a *controller*: a system component responsible for making decisions and taking actions based on the system's state. A controller is represented as mapping the system's current state to an action, which can be interpreted as a control input or a setpoint. Mathematically, we use an FSA $\langle \Sigma, \Gamma, Q, q_0, \delta, \omega \rangle$ to represent the controller, whose input alphabet Σ indicates all possible observations of the environment and output alphabet Γ indicates all possible actions. We additionally allow for a "no operation" action $\epsilon \in \Gamma$.

The controller's goal is to adjust the control input so that the system's state evolves in a way that satisfies externally provided requirements or properties. These requirements or properties are often specified using formal languages, such as *linear temporal logic* (LTL) [1].

A *model* is a transition system that may represent either the dynamics of the task environment or knowledge from other independent sources. A *model* $\mathcal{M} := \langle \Sigma_{\mathcal{M}}, \Gamma_{\mathcal{M}}, Q_{\mathcal{M}}, \delta_{\mathcal{M}}, \omega_{\mathcal{M}} \rangle$ consists of input alphabet $\Sigma_{\mathcal{M}} := 2^{P_{\mathcal{M}}}$ is a set of input symbols, where $P_{\mathcal{M}}$ is defined as the actions. $Q_{\mathcal{M}}$ is a finite set of states, $\delta_{\mathcal{M}} : Q_{\mathcal{M}} \times \Sigma_{\mathcal{M}} \times Q_{\mathcal{M}} \to \{0, 1\}$ is a non-deterministic transition function, and $\omega_{\mathcal{M}} : Q_{\mathcal{M}} \to \Gamma_{\mathcal{M}}$ is a labeling function, where $\Gamma_{\mathcal{M}} = 2^{\overline{P}}$ and $\overline{P}$ is a set of atomic propositions representing conditions of the environment.

The Planning Domain Definition Language. A *Planning Domain Definition Language* (PDDL) [9] is a formal language used in artificial intelligence and automated planning to define a planning problem. We use PDDL to describe the possible initial states of a problem, the desired goal, and the actions that can be taken to transform the initial state into the goal state. PDDL provides a standardized syntax for specifying a set of predicates—atomic propositions—describing the states of the task, the actions, and the goal specification.

Each action a in PDDL has a name, a *precondition* that must be satisfied before the action can be performed, and an of *effect* that describes how the state of the environment will change after the action is performed. The preconditions and effects are expressed as sets of atomic propositions.

4 TASK CONTROLLER CONSTRUCTION

We develop an algorithm, `LLM2Automata`, that takes a brief task description in textual form from the task designer and returns an FSA representing the task controller that can be verified against the specifications given by the task designer.

The algorithm `LLM2Automata` takes a brief text description of a task and constructs an FSA to represent the controller of the given task. Specifically, the algorithm sends the text description as the input prompt (in blue) to a GLM and obtains the GLM's response (in red), which is a list of steps for achieving the task in textual form:

```
Steps for task description
step_number_1. step description
step_number_2. step description
...
```

The algorithm uses the semantic parsing method introduced in GLM2FSA [36] to parse each step description into *verb phrases (VP)* and connective keywords. A list of pre-defined keywords is provided in Table 1. A verb phrase consists of a verb and its noun dependencies. Each step corresponds to a state in the FSA. Meanwhile, each verb phrase VP in the step description represents an *action*, and the algorithm queries the GLM to extract the precondition and effect of this action in the form of PDDL:

```
Define an action "action name" in PDDL
 Action: action name
 Precondition: a set of propositions
 Effect: a set of propositions
```

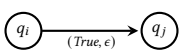
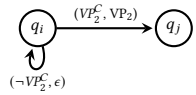
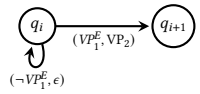
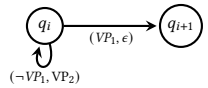
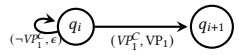
Grammar	Formal Representation	Example
VP_1 **and** VP_2	$\text{VP}_1 \wedge \text{VP}_2$	[green light] [and] [no car]
VP_1 **or** VP_2	$\text{VP}_1 \vee \text{VP}_2$	[traffic light] [or] [crosswalk]
no/not VP_1	$\neg\ \text{VP}_1$	[no] [car]
VP_1 **[step j]**		[go to step] [1]
if VP_1, VP_2. VP_2 **if** VP_1		[if] [green light], [cross]
wait VP_1 VP_2 VP_2 **after** VP_1		[wait] [green light] [cross]
VP_2 **until** VP_1		[not cross] [until] [green light]
VP_1		[cross road]

Table 1: Rules to convert natural language grammar to formal representations (propositions or FSA transitions). The keywords that define the grammar are in bold.

We use the extracted verb phrase VP_i to define the action name, and we use VP_i^C and VP_i^E to denote the precondition and effect of the action, respectively. Then, the algorithm follows the rules illustrated in Table 1 to transform natural language into propositions or automaton transitions. Each step description is translated into a state in the FSA and a set of outgoing transitions from this state.

We note that in contrast to the GLM2FSA algorithm presented in [36], we query the GLM for the preconditions and effects of each action and encode them into the constructed controller. These preconditions and effects are descriptions of the task environment prior to and after taking some actions. This explicit representation of the actions' preconditions and effects is required for the methodology we propose to ground the constructed automaton-based controllers to their task environments via image-based observations, described in Section 5.

4.1 Verifying against External Knowledge

An automaton-based model encodes the dynamics of the task environment or the task-relevant knowledge from external knowledge sources. Users can provide automaton-based models to verify whether the knowledge from the GLM is consistent with the user-provided knowledge or requirements.

Once we have the controller and the model, we use the model to formally verify whether the controller satisfies user-provided specifications. In the verification procedure, we build a product automaton $\mathfrak{P} = \mathcal{M} \otimes C$ describing the interactions of the controller C with the model $\mathcal{M}$. Then, we obtain a *specification* Φ expressed in *linear temporal logic* from the task designer or whoever wants to verify the controller. We run a model checker (e.g., NuSMV [6]) to verify if the product automaton satisfies the specification,

$$\mathcal{M} \otimes C \models \Phi. \tag{1}$$

We verify the product automaton against the specification for all the possible initial states. If the verification fails, the model checker returns a counter-example, which is a sequence of states of the product automaton $(p_1, q_1), (p_2, q_2), \ldots$ where $p_i \in Q_{\mathcal{M}}, q_i \in Q$.

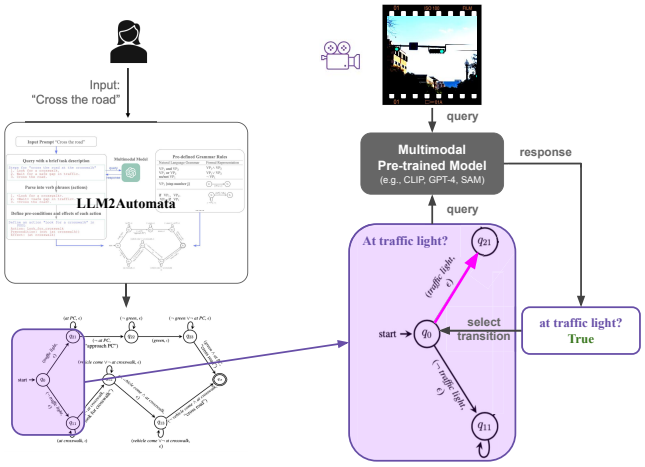


Figure 3: Demonstration of grounding the FSA-based controller to the real-world task environment through visual perceptions.

5 VERIFIABLE GROUNDING

We develop a method named `Automata2Env` to ground the controller to the real-world task environment. `Automata2Env` takes visual observations from the task environment and uses a vision-language model to determine the truth values of the atomic propositions that are relevant to the conditions specified in the controller. `Automata2Env` enables formal verification during the procedure of grounding the controller to the task environment.

5.1 The Pipeline of `Automata2Env`

To operate in the task environment, an agent starts from the initial state of the controller. The agent collects all the propositions P from the controller and gets an image observation from the task environment. It then feeds the image and all the propositions as text into a vision-language model. `Automata2Env` requires vision-language models that can output normalized scores indicating how each proposition matches the image (e.g., CLIP [24]). We refer to such scores as *confidence scores*, which are commonly provided as outputs of vision-language models. A higher score means the vision-language model is more confident that the context of the proposition is within the content of the image. We incorporate these confidence scores to approximate the perceptual uncertainties.

The overall pipeline of `Automata2Env` is as follows:

Modifying the Controller to Handle Uncertainties. We first add *uncertain* as an additional atomic proposition and modify the controller by adding a self-transition $\delta(q_i, \textit{uncertain}, q_i) = 1$ to each state q_i. Intuitively, the controller will stay in the current state, and

Algorithm 1: Proposition Evaluation under Uncertainty

procedure EvalProp(Atomic Proposition p, Observation I, Vision-Language Model **VM**, true threshold t, false threshold f) ▷ p is a string and I is an image
  score = **VM**(p, I)
  if score $\geq t$ **then return** p (p = true)
  end if
  if score $\leq f$ **then return** $\neg p$ (p = false)
  end if
  return *uncertain*
end procedure

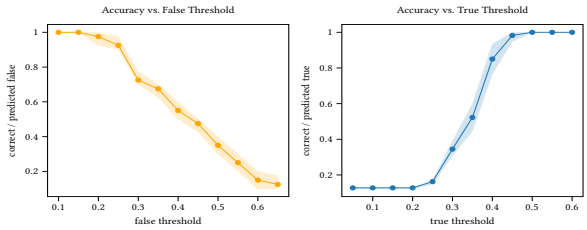


Figure 4: The left and right figures show the proposition evaluation accuracies under different false and true thresholds.

it will not perform any action if it gets uncertain observations. An example is presented in Figure 6.

Evaluating Atomic Propositions. Second, we propose an algorithm to evaluate the truth values of propositions in image observations. The algorithm takes an atomic proposition in textual form, a vision-language model that can return confidence scores and numerical thresholds as inputs. Recall that an input symbol is a set of atomic propositions. As opposed to ordinary binary evaluation, the algorithm evaluates an atomic proposition and assigns one of the three values: true, false, and uncertain. Algorithm 1 shows how we evaluate the propositions using the confidence scores from the vision-language model.

Taking Actions. Third, after evaluating the set of atomic propositions, the agent chooses one transition whose input symbol (which itself is a logical formula over the atomic propositions) evaluates to true and takes corresponding actions. A demonstration of this pipeline is in Figure 1.

5.2 Determining True and False Thresholds

Selecting the Vision-Language Model. We use the current state-of-the-art vision-language model called *Grounded-Segment-Anything* (Grounded-SAM) [14, 19] to evaluate the propositions from image observations. The Grounded-SAM is an open-domain object detection model, which can take any text as input and determine whether the object or scene described in the text appears in the image. The Grounded-SAM returns a confidence score for each detected object, and the score will be zero if it does not find the object in the image.

Validating the Vision-Language Model. Once the vision-language model is selected, we validate the selected model on an externally provided dataset to determine the values of the true and false thresholds to be used in Algorithm 1. The validation procedure is based on the following assumption:

Assumption 0: Validation images are drawn from the same distribution as the images from task environments. Hence the performance of a vision-language model on validation images and task images is consistent.

Under Assumption 0, we select an image dataset, Argoverse [3], that contains driving scenes. We use the Grounded-SAM to detect driving-relevant objects (e.g., crosswalks, traffic lights, cars) and check whether the detection results are correct. We collect the confidence scores of the detection results from Grounded-SAM and the corresponding ground truth labels from the dataset.

Determining Thresholds. We plot figures on false/true thresholds vs. proposition evaluation accuracy and present them in Figure 4. For each true threshold t, we evaluate all the detection results whose confidence scores are greater than t to true and compute the percentage of "the number of results correctly evaluated to true divided by the number of results that were evaluated to true." We denote this percentage as *proposition evaluation accuracy*. Similarly, we compute the percentage of the number of results that are correctly predicted to be false, divided by the total number of results evaluated to be false. We expect the proposition evaluation accuracies for both true and false to equal 1, which means everything that is evaluated to be true or false is correct. We denote this scenario as another assumption:

Assumption 1 ($\mathcal{A}_1$): If a proposition is NOT evaluated to *uncertain*, then the proposition evaluation is correct.

Now we can determine a true threshold t and a false threshold f according to the empirical accuracies plotted in Figure 4. Note that regardless of what thresholds we choose, this empirical estimate of the proposition evaluation accuracy is not 1. Hence we can obtain a probability of Assumption 1 being held ($\mathbb{P}[\mathcal{A}_1 = true]$) through Theorem 1.

Theorem 1. Let $\mathbf{p}_t$ be the proposition evaluation accuracy for the true threshold t and $\mathbf{p}_f$ be the accuracy for the false threshold f. Then, for each proposition evaluation whose result is not *uncertain*,

$$\mathbb{P}[\mathcal{A}_1 = true] \geq \min(\mathbf{p}_t, \mathbf{p}_f). \tag{2}$$

5.3 Verification

We now have the modified controller $\overline{C}$ that takes uncertainties into consideration and the selected thresholds. Given a model $\mathcal{M}$ and specifications Φ, we can apply model checking to verify whether the controller, when implemented in the model, satisfies the specifications during the grounding procedure. However, we need to additionally consider the perceptual uncertainties under the selected thresholds. Instead of Equation 1, we apply the model checker to verify

$$\mathcal{M} \otimes \overline{C} \models (\mathcal{A}_1 \implies \Phi). \tag{3}$$

This model-checking procedure ensures the controller satisfies the specifications, given that we captured all the perceptual uncertainties (i.e., Assumption 1 holds). If Equation 3 passes the model-checking procedure, the probability of Φ being satisfied is purely based on the degree of perceptual uncertainties, which is the probability of Assumption 1 holds. Hence we can derive a new theorem.

Theorem 2. Let event $\tilde{e} = \mathcal{M} \otimes \overline{C} \models (\mathcal{A}_1 \implies \Phi)$, let N_{max} be a user-specified parameter representing the maximum number of proposition evaluations that the controller makes during a single run. If $\tilde{e}$ is true, then the following inequality holds

$$\mathbb{P}[\mathcal{M} \otimes \overline{C} \models \Phi] \geq \mathbb{P}[\tilde{e}] \cdot \mathbb{P}[\mathcal{A}_1 = true]^{N_{max}} \geq \min\left(\mathbf{p}_t, \mathbf{p}_f\right)^{N_{max}}.$$

Note that we can only approximate $\mathbf{p}_t$ and $\mathbf{p}_f$ through empirical analysis (e.g., Figure 4). We denote the approximations of $\mathbf{p}_t$ and $\mathbf{p}_f$ as $\overline{p}_t$ and $\overline{p}_f$.

6 EMPIRICAL DEMONSTRATION

We illustrate the `LLM2Automata` algorithm and the grounding method `Automata2Env` with proof-of-concept examples on different domains. We also collect image observations from real-world environments to demonstrate the controller's behaviors in realistic deployment settings. We use the current state-of-the-art multimodal pretrained models GPT-4 [23] and Grounded-SAM [14, 19] to produce all the results in this section.

6.1 Crossing Road Example

We start the demonstration on a daily life task: cross the road. In this example, we construct a controller that can handle the crossing-road task at the traffic light.

Controller Construction. First, we query GPT-4 for the steps of crossing the road at the crosswalk at a traffic light and obtain a list of steps in textual form:

```
Steps for "cross the road at a traffic light"
 1. Approach the pedestrian crossing.
 2. Wait for the traffic light to turn green.
 3. Cross the road.
```

Next, we query GPT-4 again to define the verb phrases in PDDL and extract the preconditions and effects:

```
Define an action "approach pedestrian
    crossing" in PDDL
Action: Approach_pedestrian_crossing
Precondition: (not (at_pedestrian_crossing))
Effect: (at_pedestrian_crossing)

Define an action "Traffic light turn green"
    in PDDL
Action: Traffic_light_turn_green
Precondition: (not (traffic_light_is_green))
Effect: (traffic_light_is_green)

Define an action "cross road" in PDDL
Action: Cross_road
Precondition: (traffic_light_is_green) (
    at_pedestrian_crossing)
Effect: (at other_side)
```

After we have the verb phrases with preconditions and effects in textual form, we follow the grammar in Table 1 to transform each step into a state and its outgoing transitions. We get an FSA that represents the controller by connecting all the states with the transitions, as presented in Figure 6.

Grounding and Verification. We use the Grounded-SAM to evaluate the input symbols and implement the control logic in the real-world task environment. The Grounded-SAM takes an image and a set of propositions in textual form as inputs and classifies which propositions match the image. A proposition matches an image if the object or scenario described by the proposition appears in the image.

In the grounding procedure, we apply Algorithm 1 with a true threshold $t = 0.45$ and a false threshold $f = 0.2$. A proposition will be evaluated as *uncertain* if the score is between 0.2 and 0.45. Under these thresholds, we have $\overline{p}_t = 0.983$, $\overline{p}_f = 0.975$, and $\mathbb{P}[\mathcal{A}_1 = true] = 0.975$ according to Figure 4 and Theorem 1.

We also adjust the controller to adapt to the real-world environment with perceptual uncertainties, as presented in the bottom figure of Figure 6.

Figure 5 shows an example of grounding the controller to the real-world environment with perceptual uncertainties. We highlight the second image from the left in the observation sequence in Figure 5. Due to a confidence score of 0.4, the proposition "traffic light is green" is evaluated to *uncertain*, which triggers a self-transition at state q_2, and no action is taken. Note that the Grounded-SAM misclassified the red light to the green light. If we do not consider perceptual uncertainties, the cross-road action may be triggered at the red light.

We use the model in Figure 7 to verify the controller with uncertainties against the specification $\Phi = \neg(crossroad \wedge \neg green)$ to ensure safety. The controller satisfies the specification if Assumption 1 holds: $\mathcal{M} \otimes C \models (\mathcal{A}_1 \implies \Phi)$. Then, according to Theorem 2, the probability of the controller "never performing the cross-road action when the traffic light is not green" is at least 0.975^N. Note that N is the number of proposition evaluations that are certain during the grounding procedure ($N = 5$ in the example in Figure 5. Hence we have provided a guarantee to the safety of the controller.

6.2 Robot Arm Manipulation

We follow `LLM2Automata` to construct a controller for the task "use a robot arm to remove all the red blocks off the table." We show how we use the Grounded-SAM to perceive the operating environment and make decisions accordingly.

Controller Construction. In this example, we assume the user has some prior knowledge of the task, such as some basic knowledge of the environment and the admissible actions of the robot arm. The user thus queries GPT-4 with the following prompt:

```
 Task: place all the red blocks off the table.
Environment: there are unknown numbers of red
    blocks and yellow blocks on the table
    initially. Someone may randomly add a red
    block or yellow block to the table.
Steps for achieving the task:
 1. Target one block on the table.
 2. Classify the color of the targeted block.
 3. If the block is red, place it from the
    table to an off-table location (B). If the
     block is yellow, leave it on the table.
 4. Go to step 1.

 1. Define an action "target one block" in
    PDDL.
 Action: target-one-block
 Parameters: ()
 Precondition: (block_on_table)
 Effect: (and (block_targeted))

 2. Define an action "classify the color of
    the targeted block" in PDDL.
 Action: classify-color
 Parameters: ()
 Precondition: (block_targeted)
```

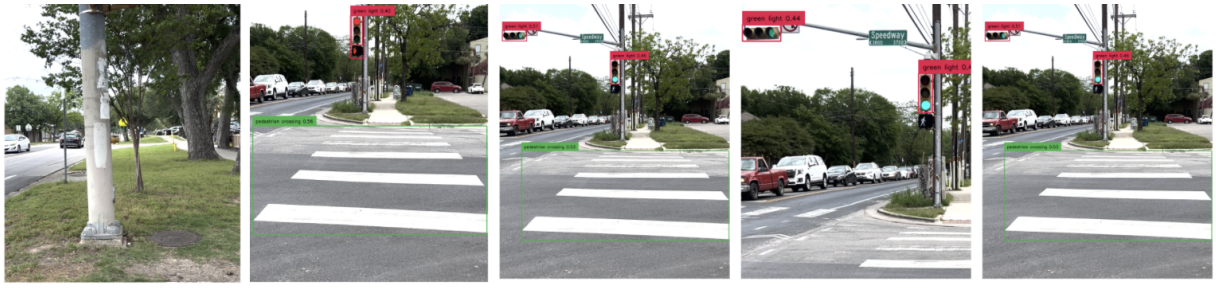


Figure 5: A demonstration `Automata2Env` implementing control logic under perceptual uncertainties. The figure shows a sequence of observations from the real-world environment, where red and green boxes with confidence scores above are the object detection results from the Grounded-SAM. We use the Grounded-SAM to measure the confidence of image content and evaluate the propositions from the controller. The resulting controller's state transitions are $q_1 \to q_2 \to q_2 \to q_3 \to q_3 \to q_4$.

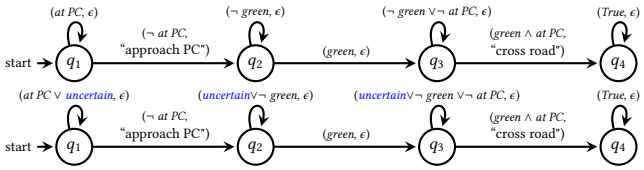


Figure 6: Automata-based representation for task "crossing the road." The top figure shows the automaton constructed from the algorithm `LLM2Automata`, and the bottom figure shows the modified automaton for grounding purposes. "PC" stands for the proposition "at pedestrian crossing" and "green" stands for the proposition "traffic light is green."

```
Effect: (or (block_red) (block_yellow))

3. Define an action "place the block" in PDDL
   .
Action: place-block
Parameters: ()
Precondition: (and (block_targeted) (
   block_red))
Effect: (not (block_targeted))

3. Define an action "leave the block" in PDDL
   .
Action: leave-block
Parameters: (?b - block ?t - table)
Precondition: (and (block_targeted) (
   block_yellow))
Effect: (not (block_targeted))
```

We present the constructed controller in Figure 9.

Grounding and Verification. Next, we verify the controller in Figure 9. Suppose a model $\mathcal{M}$ is provided from some independent knowledge source and presented in Figure 10. We want to guarantee the robot arm never accidentally places a yellow block outside the table. Hence we define the temporal logic specification

$$\Phi = \neg place \wedge yellow.$$

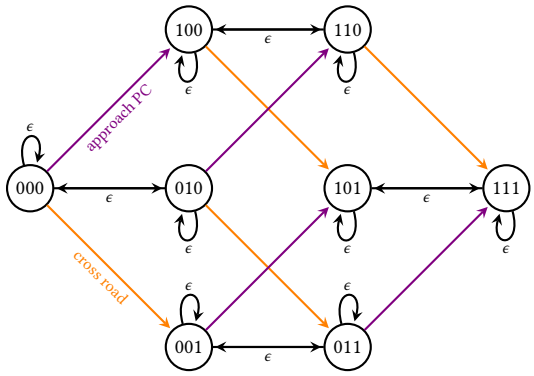


Figure 7: Transition system that represents the environment of the "crossing road at the traffic light" task. Transitions in violet and orange represent the transitions with actions "approach traffic light" and "cross-road," respectively. The label on the state indicates the value of propositions ("at PC," "green," "at other side"). For instance, 000 indicates $\neg$*at PC* $\wedge$ $\neg$*green* $\wedge$ $\neg$*at other side* and 010 indicates $\neg$*at PC* $\wedge$ *green* $\wedge$ $\neg$*at other side.*

Recall that the trajectory is defined over the union of the set of actions and the set of effects. We use this model to verify that the robot arm controller satisfies the specification Φ, under Assumption 1. The model-checking result indicates that the controller, when implemented in the model, satisfies Φ given Assumption 1 holds.

We again use the Grounded-SAM as the perception model to ground the controller from Figure 9 to the operating environment. We set the true threshold and false threshold in Algorithm 1 to 0.45 and 0.2, respectively. Therefore, the probability $\mathbb{P}[\mathcal{A}_1 = true]$ is 0.975. Figure 8 shows a full iteration of the controller ($q_1 \to q_2 \to q_3 \to q_4$). In the example from Figure 8, the probability of Assumption 1 always holding is 0.975^3. Therefore, the probability of Φ being satisfied in this example is also 0.975^3.

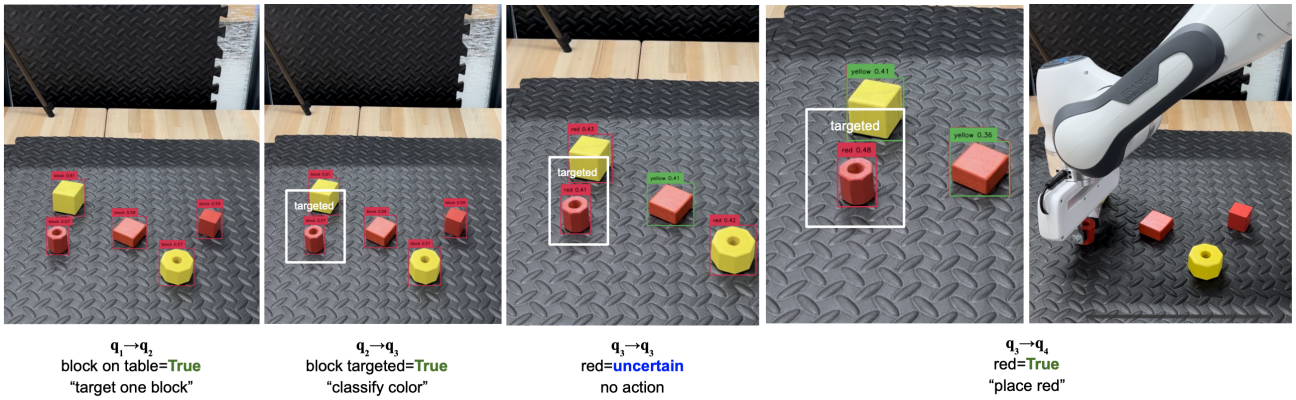


Figure 8: An example of using a robot arm to remove red blocks on the table. The figures show the object detection results from the Grounded-SAM in red and green boxes. We show the proposition evaluation results and list the state transitions and actions that are taken under the evaluated propositions.

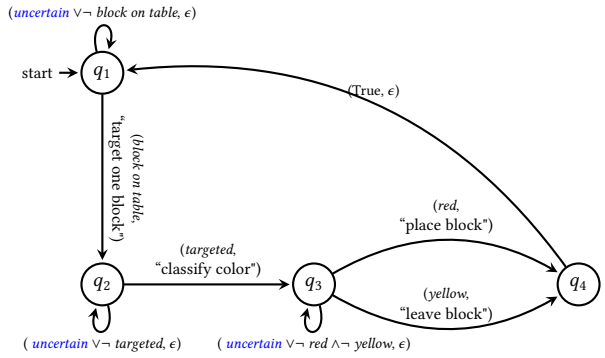


Figure 9: The controller for a task "using a robot arm to remove all the red blocks on the table" with the consideration of perceptual uncertainties.

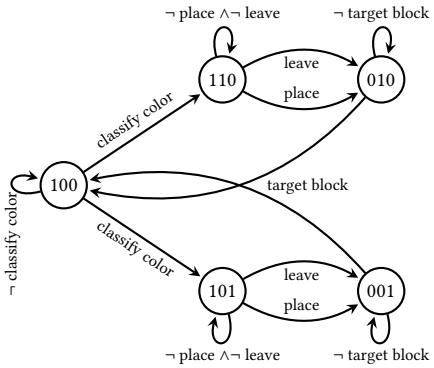


Figure 10: Transition system that represents the environment of the robot arm task. The environment requires the agent to target a block before classifying its color and to either place or leave the block only after the color is classified. The label on the state indicates the value of propositions ("block targeted," "red," "yellow").

7 CONCLUSION

We provide a proof-of-concept for the automatic construction of an automaton-based task controller of task knowledge from GLMs and the grounding of the controller to physical task environments. We propose an algorithm named `LLM2Automata` that fills the gap between the textual outputs of generative models and sequential decision-making in the aspects of synthesis, verification, grounding, and perception. The algorithm synthesizes automaton-based controllers from the text-based descriptions of task-relevant knowledge that are obtained from a GLM. Such automaton-based controllers can be verified against user-provided specifications over models representing the task environments or task knowledge from other independent sources. Additionally, we develop a grounding method `Automata2Env` that grounds the automaton-based controllers to physical environments. It uses vision-language models to interpret visual observations from the task environment and implements control logic based on the observations. `Automata2Env` utilizes the confidence scores returned by the vision-language models to ensure safety under perceptual uncertainties. Experimental results demonstrate the capabilities of `LLM2Automata` and `Automata2Env` on synthesis, verification, grounding, and perception.

Future Directions. We have developed the algorithm to create formal representations of textual task knowledge and to ground those abstract representations in the physical environment through visual perceptions. As one future direction, we can develop an active perception method to actively search for the desired objects rather than having a fixed-angle camera.

ACKNOWLEDGMENTS

This research was supported in part by the Office of Naval Research (ONR) under Grant N00014-22-1-2254 and in part by the National Science Foundation (NSF) under Grants NSF 1652113 and NSF 2211432.

REFERENCES

[1] Oliver Biggar and Mohammad Zamani. 2020. A Framework for Formal Verification of Behavior Trees with Linear Temporal Logic. *IEEE Robotics and Automation Letters* 5, 2 (2020), 2341–2348. https://doi.org/10.1109/LRA.2020.2970634

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-shot Learners. *Advances in Neural Information Processing Systems* 33 (2020), 1877–1901.

[3] Ming-Fang Chang, John Lambert, Patsorn Sangkloy, Jagjeet Singh, Slawomir Bak, Andrew Hartnett, De Wang, Peter Carr, Simon Lucey, Deva Ramanan, and James Hays. 2019. Argoverse: 3D Tracking and Forecasting With Rich Maps. In *IEEE Conference on Computer Vision and Pattern Recognition*. Computer Vision Foundation / IEEE, Long Beach, CA, USA, 8748–8757.

[4] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, and et al. 2021. Evaluating Large Language Models Trained on Code.

[5] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, and et al. 2022. PaLM: Scaling Language Modeling with Pathways.

[6] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. 2002. NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In *Computer Aided Verification (Lecture Notes in Computer Science, Vol. 2404)*. Springer, 359–364. https://doi.org/10.1007/3-540-45657-0_29

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, Minneapolis, MN, USA, 4171–4186.

[8] Xiuye Gu, Tsung-Yi Lin, Weicheng Kuo, and Yin Cui. 2022. Open-vocabulary Object Detection via Vision and Language Knowledge Distillation. In *International Conference on Learning Representations*. OpenReview.net, Virtual, 1–20.

[9] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, and Christian Muise. 2019. *An Introduction to the Planning Domain Definition Language*. Morgan & Claypool Publishers.

[10] Mutian He, Tianqing Fang, Weiqi Wang, and Yangqiu Song. 2022. Acquiring and Modelling Abstract Commonsense Knowledge via Conceptualization.

[11] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 162)*. PMLR, Baltimore, Maryland, USA, 9118–9147.

[12] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Tomas Jackson, Noah Brown, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. 2022. Inner Monologue: Embodied Reasoning through Planning with Language Models. In *Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 205)*. PLMR, Auckland, New Zealand, 1769–1782.

[13] Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. In *Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 205)*. PLMR, Auckland, New Zealand, 287–318.

[14] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. 2023. Segment Anything.

[15] Liunian Harold Li, Pengchuan Zhang, Haotian Zhang, Jianwei Yang, Chunyuan Li, Yiwu Zhong, Lijuan Wang, Lu Yuan, Lei Zhang, Jenq-Neng Hwang, Kai-Wei Chang, and Jianfeng Gao. 2022. Grounded Language-Image Pre-training. In *Conference on Computer Vision and Pattern Recognition*. IEEE, New Orleans, LA, USA, 10955–10965.

[16] Bill Yuchen Lin, Chengsong Huang, Qian Liu, Wenda Gu, Sam Sommerer, and Xiang Ren. 2023. On Grounded Planning for Embodied Tasks with Language Models. In *AAAI Conference on Artificial Intelligence*, Brian Williams, Yiling Chen, and Jennifer Neville (Eds.). AAAI Press, Washington, DC, USA, 13192–13200.

[17] Kevin Lin, Christopher Agia, Toki Migimatsu, Marco Pavone, and Jeannette Bohg. 2023. Text2Motion: From Natural Language Instructions to Feasible Plans.

[18] B. Liu, Yuqian Jiang, Xiaohan Zhang, Qian Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency.

[19] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. 2023. Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection.

[20] Yujie Lu, Weixi Feng, Wanrong Zhu, Wenda Xu, Xin Eric Wang, Miguel Eckstein, and William Yang Wang. 2022. Neuro-Symbolic Procedural Planning with Commonsense Prompting.

[21] Yujie Lu, Weixi Feng, Wanrong Zhu, Wenda Xu, Xin Eric Wang, Miguel Eckstein, and William Yang Wang. 2023. Neuro-Symbolic Procedural Planning with Commonsense Prompting. In *International Conference on Learning Representations*. OpenReview.net, Kigali, Rwanda, 1–34.

[22] Yujie Lu, Pan Lu, Zhiyu Chen, Wanrong Zhu, Xin Eric Wang, and William Yang Wang. 2023. Multimodal Procedural Planning via Dual Text-Image Prompting.

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

[24] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. In *International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 139)*, Marina Meila and Tong Zhang (Eds.). PMLR, Virtual, 8748–8763.

[25] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, Las Vegas, NV, USA, 779–788.

[26] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. 2017. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transaction Pattern Analysis Machine Intelligence* 39, 6 (2017), 1137–1149.

[27] Navid Rezaei and Marek Z. Reformat. 2022. Utilizing Language Models to Expand Vision-Based Commonsense Knowledge Graphs. *Symmetry* 14 (2022), 1715.

[28] Teven Le Scao, Angela Fan, Christopher Akiki, Elizabeth-Jane Pavlick, Suzana Ili'c, Daniel Hesslow, Roman Castagn'e, Alexandra Sasha Luccioni, Franccois Yvon, Matthias Gallé, and et al. 2022. BLOOM: A 176B-Parameter Open-Access Multilingual Language Model.

[29] Dhruv Shah, Blazej Osinski, Brian Ichter, and Sergey Levine. 2022. LM-Nav: Robotic Navigation with Large Pre-Trained Models of Language, Vision, and Action. In *Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 205)*. PMLR, Auckland, New Zealand, 492–504.

[30] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2022. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models.

[31] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Anand Korthikanti, Elton Zhang, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. 2022. Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model.

[32] Chan Hee Song, Jiaman Wu, Clay Washington, Brian M. Sadler, Wei-Lun Chao, and Yu Su. 2022. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models.

[33] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2023. ChatGPT for Robotics: Design Principles and Model Abilities. *Microsoft Autonomous Systems and Robotics Research* 2 (2023), 20.

[34] Zihao Wang, Shaofei Cai, Anji Liu, Xiaojian Ma, and Yitao Liang. 2023. Describe, Explain, Plan and Select: Interactive Planning with Large Language Models Enables Open-World Multi-Task Agents.

[35] Peter West, Chandra Bhagavatula, Jack Hessel, Jena D. Hwang, Liwei Jiang, Ronan Le Bras, Ximing Lu, Sean Welleck, and Yejin Choi. 2022. Symbolic Knowledge Distillation: from General Language Models to Commonsense Models. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Seattle, WA, United States, 4602–4625.

[36] Yunhao Yang, Jean-Raphael Gaglione, Cyrus Neary, and Ufuk Topcu. 2022. Automaton-Based Representations of Task Knowledge from Generative Language Models.