Cognitive Framework for Preference Adaptation in Human-AI Interaction

Feiyu "Gavin" Zhu

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Advised by:
Reid Simmons

Submitted in partial fulfillment of the requirements
for the School of Computer Science Honors Research Thesis.

Keywords: Human-AI Interaction, Preference Learning, Cognitive Architectures

For my family and friends who supported me through everything.

Abstract

Previous work on preference learning focuses extensively on using rewards as proxies. Despite fitting into the reinforcement learning paradigm nicely, reward-based machine learning approaches face the difficulty of fully representing personal preferences as rewards and the challenge of updating the policy with few samples. In this study, we aim to take an alternative rule-centric approach - drawing inspiration from cognitive science and building a decision-making framework centered around production rules. The production rules in the cognitive framework are abstract, modular, interpretable, and composable - all important features in human-AI interactions. Therefore, we propose a framework that combines the general knowledge of large language models and the adaptable nature of cognitive architectures. More concretely, we formally define a cognitive architecture, show how we can bootstrap its rules with a large language model and minimal human input, collect a set of human preferences in the real world, and show how the architecture we proposed can adapt to those preferences in one shot. We hope that this work will inspire more future work in rule-centric agent policies in the future.

Acknowledgments

First and foremost I would like to thank my Advisor Prof. Reid Simmons for letting me explore the topic of my interest, asking thought-provoking questions, and reviewing my work in great detail. This thesis would not be possible without his consistent support ranging from high-level idea discussion to edits in the thesis document.

I am grateful to be part of the RASL lab and the HRI community at CMU. Special thanks to Daphne Chen for running a fun course project together, Mike Lee for providing great feedback on my presentation and poster, Michelle Zhao for spending time proofreading my writing, Fern Limprayoon for the discussion at the early stage of my thesis work, Pat Callaghan for helping with my graduate school application, and Arnav Mahajan for introducing me to human-AI collaboration.

Many other faculties have also contributed profoundly to my undergraduate experience. I want to thank Prof. Yonatan Bisk for all the valuable research and life advice, Prof. Alexander Mathis for allowing me to explore Switzerland and the medical applications of machine learning, Prof. Stephanie Rosenthal for being a great course instructor that I TA for, and Prof. Pat Virtue for answering random questions I have for AI.

I am also fortunate to have some incredible friends. Mike Li has been my amazing roommate who I can talk to about literally anything from commenting on arxiv papers to tedious tax deductions. Yuchen Liang lightens the mood for every hang-out session among our friends. I also won't forget how Michael Zhou saved me from running out of OpenAI credits when I needed it the most.

Special thanks to my loveliest girlfriend ever Ihita Mandal who has been incredibly understanding, cheerful, and entertaining. All of which helped me through my hardships from time and time and enabled my undergraduate degree to end on a high note. Every time that I was down Ihita would always come around and get my feet back on the ground. My life would be wildly different without her.

Last but perhaps most importantly, I am immensely thankful for my parents, who have provided me with unconditional support even if they are on the other side of the earth. Knowing that I will always have them on my side gives me the courage to pursue my interests instead of blindly following the trend. To mom and dad, thank you for everything.

Funding

This work was partly supported by the Summer Undergraduate Research Fellowship from CMU and AI-CARING Institute (NSF IIS-2112633).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The pursuit of developing an assistive household robot has a long history and still remains largely unrealized [91]. At the same time, with the rise of large language models comes a surge of work in building more intelligent AI agents. However, as each household is unique and different people have different preferences, there is no one-size-fits-all solution. Thus, having agents that comply with different preferences will yield more coherent human-AI interaction than agents that fit the "average" human and environment.

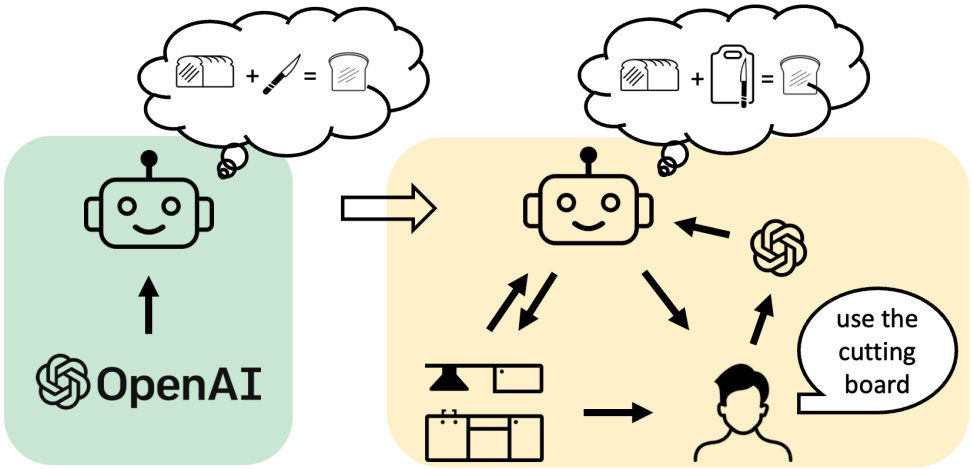


Figure 1.1: High-level interaction overview

Previous works have mainly relied on using rewards as a proxy for preferences, which often suffer from data inefficiency, misalignment issues, and other limitations [20]. More recently there has been a trend of developing generally capable agents [81] with the rise of large, foundation models. However, they aim to develop a single policy with wide capabilities instead of being optimized for being adaptable to individual needs.

In this work, we aim to develop an agent policy that is generally capable of doing each task and can perform quick and persistent adaptation to new preference requests. The general overview is shown in Figure 1.1 where the robot first bootstraps some skills in a weakly supervised fashion before being deployed into people's individual environments where the robot can still continuously improve given the users' feedback. This general framework is similar to some of the existing works [31] that try to perform personalized adaptation based on existing trained models, but we are going to propose a more efficient agent framework using this paradigm.

We base our work on the main motivation: **preference should be treated as a set of fixed routines**. Building on top of this idea, we incorporate research in cognitive architecture with the empirically powerful large language models to construct an agent framework that supports quick and persistent adaptations.

Thesis Contribution

In this thesis, we present a cognitive architecture-based solution to the problem of personalization in human-AI interaction. Specifically, we first propose to formulate preferences as rules instead of rewards. We develop a bootstrapping process that can extract procedural knowledge from large language models and convert them into an agent running a custom cognitive architecture. Then we run a pilot study on collecting user preferences in the wild and show how our approach can successfully accommodate more than 80% of them with a simple one-step query. In the end, we provided a rough sketch of future work in this direction.

Chapter 2

Related Work

2.1 Human-AI Interaction

As AI technologies are becoming more ubiquitous, many research efforts explore how humans and AI should co-exist. Some topics of interest are how AI can learn more efficiently from humans [14], how human-AI teams should collaborate [24], and how to address the explainability and safety concerns during these interactions [61]. As our work focuses on preference adaptation and requires starting from a generally capable agent, we mostly review learning from humans and preference learning.

Human-in-the-loop Learning

Naive supervised machine learning separates data collection and model training [97]. By contrast, human-in-the-loop learning assumes access to a human teacher during the learning process. The human teacher can give demonstrations, critiques, comparisons, and additional annotations as feedback to aid the agent's learning [13, 23]. This is especially beneficial for many real-world reinforcement learning problems where the reward function is hard to be fully specified but instead is easier to be reconstructed by human feedback. However, most work in this direction focuses on using human feedback to learn a task in general (e.g., landing a lunar lander in a game [23]), instead of learning personal preferences.

Preference Learning

Early work in preference learning revolves around recommender systems where the system tries to learn the preference of the viewer based on their viewing history and suggests new content. Collaborative filtering [17] builds on the assumption that people who viewed the same content in the past will prefer the same content in the future. Later this has been extended to modeling the content's latent representation for representing the content [2]. With the rise of deep learning models, other works formulate the recommender system as a reinforcement learning problem where the reward is defined as the user's response to the content recommended [89].

Contemporary work investigates preference in more complex settings such as cooperative games or tasks. One of the preference learning paradigms is to generate a library of all possible human preference models and match the specific model to the observation of the human. The library could be generated by clustering on the existing human data [96], or by analyzing the environment from a game-theoretic point of view [76]. Although these show good results in collaborative games such as Hanabi [11], they are hard to work in real-world scenarios where the preferences are more diverse.

Other works actively query the user for their preference [19, 40, 64]. Instead of trying to infer the user's preference from passively observing the user's behavior, these works actively ask questions to narrow down the user's internal reward function. Active query and feedback are essential as the user's actions don't necessarily reflect their true preference (e.g., a person can prefer a tidy room but has a messy room themselves because they are too lazy to tidy it). Therefore, in this work, we examine how the agent can adapt its policy based on active, verbal feedback.

2.2 Cognitive Architecture

Cognitive architectures are frameworks designed by the cognitive science community to model the human cognition process. Many cognitive scientists, including Allen Newell [57], believe that the human mind is like a computer with its architecture, and the skills and memories we have are like programs and data that run on this brain computer.

Alternative Approach to Deep Learning

Cognitive architectures have been one approach in the pursuit of artificial general intelligence that attempts to unify all aspects of human cognition computationally [58]. Despite the variety of architectures developed, most of them share the same central components, consisting of declarative memory reflecting knowledge of the world, procedural memory dictating the agent's behavior given certain scenarios, and short-term working memory that assists reasoning and planning [46].

The procedural memory is represented by a set of production rules, each with a precondition and an effect. Agents operate in perceive-plan-act cycles, dynamically matching relevant features of the environment to the production rules and applying their effects. Unlike operators in symbolic planning, production rules do not represent alternative actions but instead reflect different contextual knowledge [44]. These rules can be reinforced and modified throughout the agent's learning process.

We base our work on cognitive architectures mainly because they are abstract and modular. The production rules are abstract as they contain variable placeholders that dynamically bind to the environment. This allows the agent's policy to generalize to novel environments or novel objects. For example, consider the rule "if a room is not in use and lights are on, then turn off the light". The "room" is abstract as in it can be applied to any room, even the unseen ones. With the strong generalization capability, the user only has to specify their preferences in terms of general instructions such as "put things back to where the belong" instead of specific instance-based instructions such as "put the mug in the cabinet".

On the other hand production rules are modular because each production rule only represents one step of reasoning in one task. It allows the user to update part of the agents' policy with the guarantee that the agent's behavior in other tasks won't change. This ensures the agent policy can be updated according to multiple preference specifications without suffering from catastrophic forgetting as in the deep learning approaches [52].

Development of Cognitive Architectures

Psychology-focused research aims at developing cognitive architectures using an iterative process [6]. They start with some very primitive architecture based on our understanding of the functions of different brain regions. Each function is abstracted into components in the architecture: similar to having CPU and memory hierarchy in a computer architecture, there are perception, goal, and imaginary modules in a cognitive architecture. Then experiments are conducted to compare the human performance with the

prediction of the model (program) complying with the architecture. During an experiment, behavior data, such as reaction time and task performance, and physiological data, such as functional magnetic resonance imaging (fMRI) and electroencephalogram (EEG) signals, are recorded to be compared with the estimates of the model. When there is a misalignment between human data and the estimation from the model, new features or components are introduced to the architecture to accommodate the difference [27]. A variety of tasks including classic psychology experiments [5], algebra [66], and video games [7] are used to validate and refine cognitive architectures. As a result of these developments, cognitive architectures align well with the actual functioning of human brains and are capable of hosting a wide range of tasks. There are more than 20 cognitive architectures developed and used in the research community, with subtle differences between each other [42]. The most well-known examples are ACT-R [6] and SOAR [44].

Despite some pioneering work on data-driven cognitive model creation [30], almost all previous work generate their initial set of production rules manually, limiting their application to simple environments such as blocks world or psychology experiments [62]. Therefore, to alleviate the demand for human labor and make use of cognitive architectures in complex environments such as the household domain, we first develop a bootstrap process that utilizes large language models to initialize a cognitive model.

Applications of Cognitive Architectures

Interactive Task Learning [45] aims at learning the underlying concepts of a task through teacher-student interactions. Many works utilize cognitive architecture as the backbone for the student model to make use of its multiple learning mechanisms. The most distinctive difference of using a cognitive agent instead of a statistical deep learning model is that the cognitive agent is self-aware of whether it is capable of representing the instructions from the teacher. When it fails to learn a specific instruction, it can query the instructor for further clarification or rephrasing. Previous works had focused on how to efficiently use LLM as the instructor in the interactive task learning setting [37, 38].

More recently, there is a rising interest in formulating generally capable agents as a combination of cognitive architectures and large language models [77] and trying to combine them [70]. Many works acknowledge that neither the symbolic reasoning architecture nor large language models by themselves can lead to generalist agents. However, these existing works all stay at the conceptual level, while our work gives a concrete implementation.

2.3 Large Language Models

Emergent behavior has been discovered for language models where pre-trained models demonstrate a sharp and unexpected increase in performance [86]. One explanation for this phenomenon is that the human language is sparse in high dimensional space and thus becomes easier for the language models to learn when the context is longer. In addition to the language domain, similar emergence behavior has also been shown in the vision domain [36] where the model can segment unseen and irregular objects such as dough or even shadow. Many current works are trying to explore the emergent behavior in the manipulation [34] and science domain [15]. Note that we use large language models and foundation models interchangeably.

In this work, instead of trying to build a new large foundation model or using it as a backbone for decision-making, we make use of the existing large language model to 1) replace human labor by providing knowledge of the world, and 2) bridge natural language and executable code.

Prompting Techniques

Training a large language model could be extremely costly, therefore many other works explore how to get the most out of a pre-trained model in a zero-shot fashion by strategically prompting the model. We make use of these prompting techniques to extract general knowledge of the world when bootstrapping our cognitive agent.

Chain-of-thought [87] prompts the large language model to generate a response step-by-step, breaking a complex problem into subparts. This often leads to better results as the large language model can be seen as a noisy knowledge retriever and the easier the (sub)problem the better its result. Later work extends this to a search tree [90]. Meta prompting [80] makes use of system prompts to assign different roles to the same large language model. There is a conductor model that interacts with the human and synthesizes information from other expert or critic models.

Foundation Models for Embodied Agents

It has been well-acknowledged that large language models by themselves have limited capabilities when deployed to embodied tasks due to their limited context length, having different embodiment models, etc. Therefore, many works explore combining them with other components to create an embodied agent.

There are several popular schemes for integrating large language models with other components. The most common one is to have a separate memory module to regulate prompting [62, 93]. The prompts depend on the memory of the past or other factual knowledge, conditioning the content generation of the large language models. This allow the model to have more relevant information in its limited context window. Work in this direction can be seen as retrieval augmented generation [28].

Other work has looked at post-processing the output of the large language models to ensure the actions they propose respect the affordance of the embodied agent. It could be a separate, learned affordance model [3], or simply just another large language model with a different prompt as a critic [85]. As most large language models are trained using human data, they assume human affordance (e.g., two hands, bipedal, etc.). These integration techniques aim to reshape the action to fit the specific task environments.

Another popular approach is to combine the code generation capability of large language models and existing compilers or domain-specified language [78, 84, 94, 95]. These approaches use code to represent a plan for solving the current task and make up for the weak reasoning capabilities of the large language models. Other work extended this to use other tools such as existing deep learning models [63]. Our work is most similar to this approach where we use large language models to generate code running in a cognitive architecture. However, unlike the situation-grounded code produced by these methods, our approach generates parameterized productions with learnable weights. This allows more generalization capabilities and choosing the best plan among multiple applicable plans.

Concerns Regarding Large Language Models

The most well-acknowledged issue is hallucination [32] where the language model generates factually false or illogical responses. This is because large language models are fundamentally statistical models so there is no mechanism to ensure the soundness of the output. Because of this, we introduce post-processing techniques during the bootstrapping process to ensure the production rules generated by the large language models are desirable.

Other works have shown data leak [56] and revealing unethical information [99], which all suggest that an agent's decision-making cannot rely entirely on large language models but have to be combined with some other components, such as a cognitive architecture.

Chapter 3

Lixr Framework

This chapter describes the mathematical formulation of the proposed Lixr[1] framework, and give a high-level overview of its working process. It is heavily inspired by existing cognitive architectures, especially SOAR [44] and ACT-R [6]. The main conceptual difference lies in alleviating many of the constraints imposed by human cognition (e.g., need explicit retrieval to long-term memory) and the main implementation difference is the choice of using Python as the backend for more expressive precondition functions and better code generation from existing large language models.

3.1 Formulation

We use Σ^* to represent the set of all possible *value encodings*. Note that the empty string is also part of this set: $\varepsilon \in \Sigma^*$. We also assume time is discrete and can be represented as $\mathbb{Z}$ where 0 is present, negative numbers are in the past, and positive numbers are in the future [2].

Variable Timeline

We define a *variable timeline* as $T : \mathbb{Z} \to \Sigma^*$, which function that maps some *time* to some *value*. Conceptually each timeline is a function that records the value of a variable in the past and gives some prediction of the future. Let T_ε be the *empty timeline* that is $\forall t, T_\varepsilon(t) = \varepsilon$. Let the set containing all timelines be $\mathbb{T}$.

We define two higher-order function interfaces for timelines. An *advance* function:

$$Adv(T)(t) = T(t+1) \qquad \forall t \in \mathbb{Z} \tag{3.1}$$

that represents moving the timeline forward for one step in time. And a *compose* function that merges two timelines (whose implementation depends on the internal representation of the timelines):

$$T = T_1 \circ T_2 \tag{3.2}$$

In this work, each timeline is implemented as a dictionary that maps time steps to values. When querying for the value of the variable at some given time, it returns the value at the closest time less than

[1] Language-instructed and executable rules.

[2] Note that this is different from the actual implementation where the agent keeps track of the absolute time (i.e., 0 represents the time when the agent was boot up instead of present). This chapter uses 0 to describe the present to avoid keeping track of the current time as part of the agent, which is practically useful but conceptually redundant.

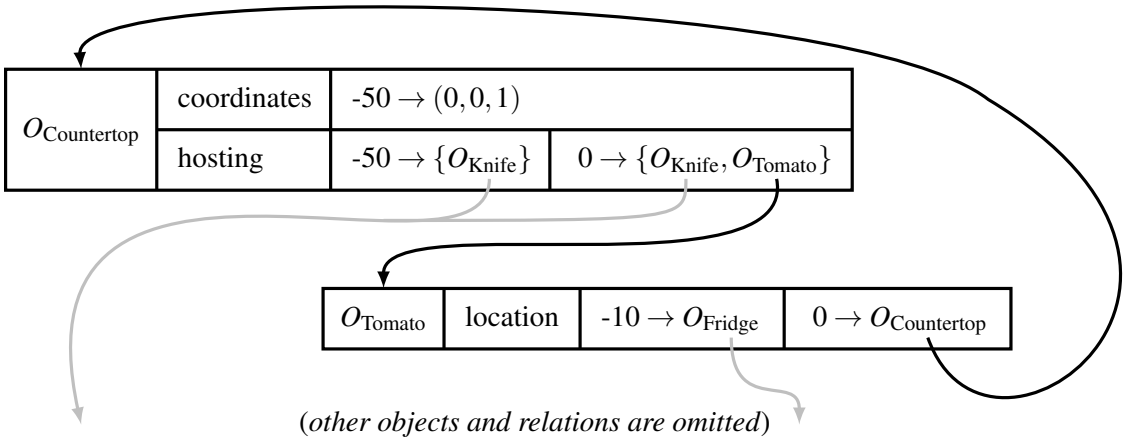


Figure 3.1: Object representation with timelines. It shows that 50 time steps ago only the knife was on the countertop. 10 time steps ago the tomato is found in the fridge. And at present the tomato is moved to the countertop. This makes it easy to reason about the past (e.g., where the tomato was 5 steps ago).

or equal to the query. For example, we can represent "the location of the robot" as:

$$T_{\text{location}} = \{\text{-10} \rightarrow \text{“fridge"},\ \text{-1} \rightarrow \text{“stove"},\ 0 \rightarrow \text{“sink"}\} \tag{3.3}$$

which means the robot was at the fridge 10 time steps ago, moved to the stove 1 time step ago, and is currently at the sink. If we want to know where the robot is at 5 time steps ago, we will get

$$T_{\text{location}}(\text{-5}) = T_{\text{location}}(\text{-10}) = \text{“fridge"} \tag{3.4}$$

Using this definition, composing two timelines is the same as taking the union of the two corresponding dictionaries. If there is a conflict between the two (i.e., having different values for the same timestamp), then the value of the timeline on the right hand side is taken.

For the sake of simplicity, we use the following to denote a constant timeline

$$T_{\text{brand}} = \{\text{“Hello Robot Stretch"}\} \tag{3.5}$$

This can be extended to incorporate other interpolation or extrapolation functions, but the concept remains the same and we leave that to future work. For the rest of the section, we will use the dictionary notation as shown in Equation 3.3.

Object

Building on top of timelines, we can represent an *object* as $O : \Sigma^* \rightarrow \mathcal{T}$ which maps attribute names to their timeline. For a value $v \in \Sigma^*$ that is not an attribute of the object O, we let $O(v) = T_\varepsilon$. Let the set of all objects be $\mathbb{O}$. Representing each object as a set of attribute timelines makes it easier to reason about the past and future state of the object.

We assume access to an ID function $ID : \mathbb{O} \rightarrow \mathbb{Z}$ that assigns each object instance with a unique ID. This function exists because the objects are encodable. Note that $\mathbb{O} \subseteq \Sigma^*$, and an object can be a value in

a timeline. Also similar to the timelines, objects can be advanced:

$$Adv(O)(v) = Adv(O(v)) \qquad \forall O \in \mathbb{O}, v \in \Sigma^* \tag{3.6}$$

We can represent a variety of instances as an object. Most naively we can represent the *atomic objects* observed in the world (i.e. the objects detected by a general purpose object detector [36]). For example, we can represent a plate in the kitchen as:

$$O_{\text{plate}} = \begin{cases} \text{“color"} & \to \{\text{-50} \to \text{“white"}\} \\ \text{“carrying"} & \to \{\text{-50} \to \varepsilon,\ \text{-20} \to O_{\text{apple}}\} \end{cases}$$

Note that we assume access to ground truth information of the relationship between objects (e.g., “apple is on the plate"), which can be obtained by scene graph generation models such as [92].

Since objects can be values in the timeline, we can also represent *reference objects* that are not technically observed from the perception but are useful to reason about conceptually. For example, we can represent a utensil drawer by:

$$O_{\text{utensil drawer}} = \begin{cases} \text{“name"} & \to \{\text{“utensil drawer"}\} \\ \text{“drawer"} & \to \{O_{\text{drawer}}\} \end{cases}$$

Where O_{drawer} is the actual drawer segmented from the perception, but the utensil drawer is just a reference wrapper that makes further reasoning and policy transfer easier.

We can also exploit the object structure and use it to represent a *goal*. For example, the pick and place goal can be specified as:

$$G_{\text{pick and place}} = \begin{cases} \text{“type"} & \to \{\text{“pick and place"}\} \\ \text{“target"} & \to \{O_{\text{fork}}\} \\ \text{“location"} & \to \{O_{\text{drawer}}\} \\ \text{“condition"} & \to \{(O_{\text{fork}}(\text{“location"})(0) = O_{\text{drawer}})\} \end{cases}$$

Let the set of all goals be $\mathbb{O}_G \subseteq \mathbb{O}$. Within $\mathbb{O}_G$, we assume that the agent already comes with some basic action primitives that it knows how to solve. Let the set of atomic actions be $\mathbb{O}_A \subseteq \mathbb{O}_G$. Similar to having an empty string, we can also have an “empty action" $A_\varepsilon \in \mathbb{O}_A$ that represents the “do nothing" action. Also let $\mathbb{O}_{\overline{G}} = \mathbb{O} \setminus \mathbb{O}_G$ to represent the set of non-goal objects.

Object Snapshot

An object snapshot $O' : \Sigma^* \to \Sigma^*$ represents the attribute values of an object at a specific time step. It can be seen as an observation of an object. Therefore, its range is not timelines of attribute values but the values themselves. Let $\mathbb{O}'$ denote the set of all object snapshots.

We also assume access to an ID function for object snapshots ($ID' : \mathbb{O}' \to \mathbb{Z}$) that can associate an object snapshot with the ID of the object it belongs to. This is used to associate new observations with existing objects. We can define the compose function between an object O and its corresponding snapshot O' as:

$$(O \circ O')(v) = \left\{O(v) \circ \{0 \to O'(v)\} : v \in \Sigma^*\right\} \tag{3.7}$$

This represents incorporating the observation O' into O to create a new object. By the definition of conflict resolution in objection composition, we always use the value of the observation when there is a conflict.

This can be used to represent the agent updating its belief. This is because having a conflict indicates the agent's prediction based on the existing object differs from the actual value from the new observation. We also define an instantiate function that converts an object snapshot into an object:

$$Inst(O')(v) = \{0 \to O'(v)\} \qquad \forall v \in \Sigma^* \tag{3.8}$$

State

We represent a *state* $S \subseteq \mathbb{O}$ as a set of objects. This is consistent with previous work in task and motion planning [54] and many object-centric approaches in perception [74]. Note that the goal objects can also be part of the state. Let $\mathbb{S}$ denote the state space (which is the power set of $\mathbb{O}$). Note that a state doesn't have to be a state of the environment. It can also represent the internal state of the agent. Again we can define the advanced function for a state as:

$$Adv(S) = \{Adv(O) \colon O \in S\}. \tag{3.9}$$

We use $S \Rightarrow \alpha$ to indicate a query α (e.g., $(O_{\text{fork}}(\text{“location"})(0) = O_{\text{drawer}})$) is true given the state.

Observation

We assume the world is partially observable so we separate the representation of an *observation* and the representation of a state. We represent an observation $S' \subseteq \mathbb{O}'$ as a set of object snapshots. Let $\mathbb{S}'$ denote the observation space (which is the power set of $\mathbb{O}'$). Similar to the composition of objects and object snapshots, a state S can incorporate a new observation S' by

$$\begin{aligned} S \circ S' = &\left\{O \circ O' \colon O' \in S' \mid \exists O \in S,\ s.t.\, ID(O) = ID'(O')\right\} && \text{(update existing objects)} \\ &\cup \left\{Inst(O') \colon O' \in S' \mid \forall O \in S,\ ID(O) \neq ID'(O')\right\} && \text{(create new objects)} \end{aligned} \tag{3.10}$$

World

We model the world as a tuple $\langle \mathcal{W}_0, \mathcal{T}, Obs \rangle$ where $\mathcal{W}_0 \in \mathbb{S}$ is the initial state of the world (unknown), $\mathcal{T} : \mathbb{S} \times \mathbb{O}_A \to \mathbb{S}$ is the (non-deterministic) state transition function (known), and $Obs : \mathbb{S} \to \mathbb{S}'$ is the observation function that emits an observation given a world state (known).

Production

A production is defined as $P = \langle V, Pre, Eff, u \rangle$. It is very similar to an operator in PDDL [26], where $V \in (\Sigma^*)^n$ is a list of (local) variable names; $Pre : \mathbb{O}^{|V|} \to \{0,1\}$ is a *precondition* function dictating whether a given variable combination passes the precondition check (i.e., whether the production is applicable); $Eff : \mathbb{O}^{|V|} \to \mathbb{O}$ defines the *effect* of the production on given the variable combinations; and $u \in \mathbb{R}$ is the estimated *utility* of applying the production. In the current formulation, both the *Pre*condition and *Eff*ect functions are deterministic. Let the set of all productions be $\mathbb{P}$. For the sake of simplicity, we use $|P|$ to denote $|V(P)|$, i.e., the number of variables involved in the precondition and effect functions.

A production is a *subgoal proposing* production if its *Eff* function returns a goal object $\mathbb{O}_G$. These productions represent context-dependent task decomposition. For example, we can have a production for

the “slicing" task as (for the sake of simplicity, we use `O.attr` to denote $O(attr)(0)$):

$$V = \langle \text{“object"}, \text{“task"} \rangle \tag{3.11}$$

$$\mathit{Pre}(\texttt{object, task}) = (\texttt{task.type} = \text{“slice"} \wedge \texttt{task.target} = \texttt{object}) \tag{3.12}$$

$$\wedge\ (\texttt{object.location} = \text{“unknown"}) \tag{3.13}$$

$$\mathit{Eff}(\texttt{object, task}) = \{\text{“type"} \rightarrow \text{“find"},\ \text{“target"} \rightarrow \texttt{object}\} \tag{3.14}$$

which represents that if the current task is to slice an object (Line 3.12) and the location of the target object is unknown (Line 3.13) then propose a subgoal for finding the corresponding object (Line 3.14). The context used in this production is that the target object is at an unknown location. Note that the *Pre* function can be *any* function, not just logical symbol testing.

On the other hand, a production is a *compositional object* production if its *Eff* function returns a non-goal object $\mathbb{O}_{\overline{G}}$. These productions represent the rules for compositional objects. For example, we can have a production for defining the “sandwich":

$$V = \langle \text{“bottom"}, \text{“stuffing"}, \text{“top"} \rangle \tag{3.15}$$

$$\mathit{Pre}(\texttt{bottom, stuffing, top}) = (\texttt{bottom.category} = \texttt{top.category} = \text{“}\mathit{bread}\text{”}) \tag{3.16}$$

$$\wedge\ (\exists O_1, O_2, \ldots, O_k,\ s.t. O_1 = \texttt{stuffing} \wedge O_k = \texttt{top} \wedge O_i.\texttt{on} = O_{i-1}) \tag{3.17}$$

$$\wedge\ (\exists O_1, O_2, \ldots, O_k,\ s.t. O_1 = \texttt{bottom} \wedge O_k = \texttt{stuffing} \wedge O_i.\texttt{on} = O_{i-1}) \tag{3.18}$$

$$\mathit{Eff}(\texttt{bottom, stuffing, top}) = \{\text{“type"} \rightarrow \text{“sandwich"},\ \text{“name"} \rightarrow \texttt{stuffing.category} + \text{“sandwich"}\} \tag{3.19}$$

which represents that if there is a stack of objects (Lines 3.17, 3.18) where the top and bottom are both bread (Line 3.16) then name the stack a sandwich by its stuffing (Line 3.19).

As shown in the previous examples a production is **abstract** in nature: the `object` could be anything in the subgoal proposing production and the “sandwich" production can represent a chicken sandwich, tuna sandwich, etc. by binding `stuffing` to different objects. This enables it to exploit the locality and sparsity [53] in the environments and support systemic generalization. Additionally, productions are also **modular** as different production defines different aspects of the policy. They are **composable** as the effect of a production can be used in another production (e.g., it can represent “slice a sandwich" by chaining the two productions above). Note that each production should only represent a single step of reasoning and it is through chaining them the agent can achieve longer horizon planning. This is important for a concise representation of the agent policy because it allows productions to be reused as much as possible. It is also easier for the large language model to generate such a production rule due to their simplicity.

Meta-Production

A production produces a new object upon application, while a *meta production* produces a context-dependent *comparison* between objects. A meta production is defined as $P' = \langle V, \mathit{Pre}, \mathit{Cmp} \rangle$ where V and *Pre* are the list of variables and the precondition function just like in a production. The $\mathit{Cmp} : \mathbb{O} \times \mathbb{O} \rightarrow \mathbb{C}$ (where $\mathbb{C}$ is $\{=, <, >\}$) is a function that takes in two objects and returns a symbolic comparison on whether one is better than another or both are equally good. We use $\mathbb{P}'$ to denote the set of all meta productions.

This can also be used to effectively *disable* some (sub)goal in the agent by specifying the "do nothing" action (A_ε) is better than the (sub)goal. For example, the following can be used to represent one should never put metal objects into the microwave:

$$\begin{aligned} V =& \langle\rangle \\ Pre() =& True \\ Cmp(\mathtt{P},\mathtt{Q}) =& \begin{cases} < & \text{if } \mathtt{P.material} = \text{“}metal\text{”} \wedge \mathtt{Q} = A_\varepsilon \\ > & \text{if } \mathtt{Q.material} = \text{“}metal\text{”} \wedge \mathtt{P} = A_\varepsilon \\ = & \text{otherwise} \end{cases} \end{aligned} \tag{3.20}$$

Support

We say a pair of object combination and a production $\langle \mathbf{v}, P \rangle$ is a *support* of an object O if

$$Pre(P)(\mathbf{v}) \wedge Eff(P)(\mathbf{v}) = O \tag{3.21}$$

Similarly, we say a pair of object combinations and a meta production $\langle \mathbf{v}, P' \rangle$ is a support of a preference $O_1 < O_2$ if

$$Pre(P')(\mathbf{v}) \wedge Cmp(P')(O_1, O_2) =' <' \tag{3.22}$$

We denote these as $\langle \mathbf{v}, P \rangle \mapsto O$ and $\langle \mathbf{v}, P' \rangle \mapsto (O_1 < O_2)$ respectively. We also define a *supported* set of objects of a state-production pair to be

$$supported(\langle S, P \rangle) = \{O \in \mathbb{O} \mid \exists \mathbf{v} \in S \; s.t. \; \langle \mathbf{v}, P \rangle \mapsto O\} \tag{3.23}$$

This concept is used when defining the environment knowledge graph in the agent.

Heuristics

The heuristics of the agent are based on the utility of its productions. When some subgoal proposing production P is applicable for a goal G and produces a new subgoal G_1, that is ($\|$ denotes concatenation)

$$\langle \mathbf{v} \parallel G, P \rangle \mapsto G_1 \qquad \text{for some } \mathbf{v} \tag{3.24}$$

then we can estimate that the agent has $u(P)$ probability of success in G if it attends to G_1 as a subgoal. Intuitively, production P represents a *strategy* for G under context $\mathbf{v}$. For example,e this can represent in order to "open a package" (G) when "the agent has no tool" ($\mathbf{v}$), a good strategy is to "find a pair of scissors" (G_1) first. In this case, the probability of "open a package" given "a pair of scissors is found" is $u(P)$. We denote this as

$$H_G(G_1 \mid \mathbf{v}, P) = u(P) \tag{3.25}$$

As we noted earlier, productions can be chained, so the probability of success can be computed by taking the product of different productions in the chain. For example, if we have two productions P_1, P_2 such that

$$\langle \mathbf{v}_1 \parallel G, P_1 \rangle \mapsto G_1 \wedge \langle \mathbf{v}_2 \parallel G_1, P_2 \rangle \mapsto G_2 \qquad \text{for some } \mathbf{v}_1, \mathbf{v}_2 \tag{3.26}$$

then we can estimate the contribution of G_2 (a second degree subgoal of G) to G as

$$H_G(G_2 \mid \mathbf{v}_1, \mathbf{v}_2, P_1, P_2) = u(P_1) \cdot u(P_2) \tag{3.27}$$

If there are no such chain between G and G_2, then we let $H_G(G_2) = 0$. And naturally $H_G(G) = 1$. If there are multiple chains, we take of sum of the heuristics to reflect that a second-degree subgoal can contribute in multiple ways. More formally if

$$\begin{aligned} &\langle \mathbf{v}_1 \parallel G, P_1 \rangle \mapsto G_1 \wedge \langle \mathbf{v}_2 \parallel G_1, P_1 \rangle \mapsto G_2 \\ \wedge\ &\langle \mathbf{w}_1 \parallel G, Q_1 \rangle \mapsto G_3 \wedge \langle \mathbf{w}_2 \parallel G_3, Q_2 \rangle \mapsto G_2 \end{aligned} \tag{3.28}$$

then

$$H_G(G_2 \mid \mathbf{v}_{1,2}, \mathbf{w}_{1,2}, P_{1,2}, Q_{1,2}) = u(P_1) \cdot u(P_2) + u(Q_1) \cdot u(Q_2) \tag{3.29}$$

For example, one strategy for opening a package (G) is to use a pair of scissors (P_1) and to find a pair of scissors the agent needs to go to the drawer (G_2). At the same time,e there is another strategy which is to use an exacto knife (Q_1), which also requires going to the drawer (G_2). Therefore going to the drawer should be a preferred action for the original goal of opening a package.

We can extend this concept to accommodate multiple goals and estimate the contribution of a single action A as

$$H_{\mathcal{G}}(A \mid \mathcal{S}, \mathcal{P}) = \sum_{G \in \mathcal{G}} H_G(A \mid \mathcal{S}, \mathcal{P}) \tag{3.30}$$

3.2 Agent

Components

An agent is defined as $\langle \mathcal{K}^E, \mathcal{K}^W, \mathcal{P}, \mathcal{P}', \gamma \rangle$. Where the $\mathcal{K}^E$ is a directed bipartite graph $\mathcal{K}^E = \langle \mathcal{V}^E, \mathcal{R}^E, \mathcal{E}^E \rangle$ that represents the *environment knowledge* where

$$\mathcal{V}^E \in \mathbb{S} \cup \mathbb{P} \cup \mathbb{P}' \tag{3.31}$$

is the set of objects, productions, and meta-productions involved in the working memory,

$$\mathcal{R}^E = \left\{ \big(\mathbf{v} \parallel P \parallel O\big) \colon P \in \mathcal{P}, \mathbf{v} \in \big(\mathcal{V}^E\big)^{|P|} \mid \langle \mathbf{v}, P \rangle \mapsto O \right\} \tag{3.32}$$

$$\cup \left\{ \big(\mathbf{v} \parallel P' \parallel C\big) \colon P' \in \mathcal{P}', \mathbf{v} \in \big(\mathcal{V}^E\big)^{|P'|} \mid \langle \mathbf{v}, P \rangle \mapsto C \right\} \tag{3.33}$$

is a set of relations between them (each relation involves one production or meta-production, a list of objects for the context, and a final outcome which can be either a new object or a comparison result). Essentially each relation tuple represents a support between (meta)productions, objects, and the outcome. And

$$\mathcal{E}^E = \{(v, r) \in \mathcal{V}^E \times \mathcal{R}^E \mid v \in r_{:\text{-}1}\} \cup \{(r, v) \in \mathcal{R}^E \times \mathcal{V}^E \mid v = r_{\text{-}1}\} \tag{3.34}$$

represents that for each relation r representing $\langle \mathbf{v}, P \rangle \mapsto O$ there are edges pointing from elements in $\mathbf{v}$ and P to r, and there is an edge pointing from r to O. Note that when P and P' are given, $\mathcal{G}$ depends only on $\mathcal{V}^E$, therefore there exists a function

$$\mathit{Expand}_{\langle \mathcal{P}, \mathcal{P}' \rangle}(\mathcal{V}^E) = \mathcal{K}^E \tag{3.35}$$

that takes the objects from the environment and *expand* it to the entire directed bipartite graph.

$\mathcal{G}^E$ is similar to the working memory in previous cognitive architecture work. It accumulates all the previous observations of the environment, keeps track of all the (sub)goals, and represents the information as a (internal) state. Unlike SOAR [44] which has a separate module for episodic memory, in this work, we take advantage of the timelines and just represent the episodic information in the working memory In addition to the objects in the environment, environment knowledge also contains the (sub)goals that the agent is attending to. Appendix B is an example of the environment knowledge in an agent.

The world knowledge $\mathcal{K}^W$ is similar to the declarative memory in ACT-R [6] that encodes the general knowledge about the world. Unlike the environment knowledge that depends on the specific environment and experience of agents, the world knowledge can be shared across all agents. The advantage of having a separate world knowledge instead of directly encoding the knowledge into production rules is to have a more condensed knowledge representation since the same piece of world knowledge can be reused by multiple production rules. The separation of world knowledge and environment knowledge is mainly for the benefit of conceptual clarity.

The procedural knowledge $\mathcal{P}$ defines what the agent will do under certain conditions and is assumed to respect the agent's affordances. The meta-production rules $\mathcal{P}'$ are mainly used for conflict resolution and we will show in a later chapter that they are useful in updating the agent policy in few-shots.

Initially $\mathcal{V}_0^E = \mathcal{K}^W \cup \mathcal{P} \cup \mathcal{P}' \cup \mathcal{G}$ represents that the only working knowledge the agent has is its predefined world and procedural knowledge (we will show in later chapter on how to acquire these) along with a set of predefined goals. And as the agent explores the environment, the agent updates its environment knowledge by

$$\mathcal{V}_{t+1}^E = Adv\left(\mathcal{V}_t^E\right) \circ Inst\left(S'_{t+1}\right) \tag{3.36}$$

$$\mathcal{K}_{t+1}^E = Expand_{\langle \mathcal{P}, \mathcal{P}' \rangle}(\mathcal{V}_{t+1}^E) \tag{3.37}$$

Action Selection

Conceptually, at each step with agent state $\langle \mathcal{K}_t^E, \mathcal{K}^W, \mathcal{P}, \mathcal{P}', \gamma \rangle$, the agent's policy is defined as

$$\pi(S'_t) = \arg\max_{A \in \mathbb{O}_A} H_{\mathcal{G}}(A \mid \mathcal{V}_t^E) \cdot Acc(A \mid \mathcal{V}_t^E) \tag{3.38}$$

where $Acc(A \mid \cdot)$ evaluates whether the action A is *acceptable* given the context. More formally:

$$\begin{aligned} Acc(A \mid \mathcal{S}, \mathcal{P}, \mathcal{P}') \Leftrightarrow \neg\big(&\exists A' \in \mathbb{O}_A, \mathbf{v} \in \mathcal{S}, P' \in \mathcal{P}', \\ &s.t.\ H_{\mathcal{G}}(A' \mid \mathcal{S}, \mathcal{P}) > 0 \ \wedge\ \langle \mathbf{v}, P' \rangle \mapsto (A < A')\big) \end{aligned} \tag{3.39}$$

In plain English, A is acceptable if and only if no other action is preferable than A given all the applicable meta-production rules. The overall policy of the agent finds the acceptable action with the best heuristic value.

Implementation-wise we can compute the heuristics of each object using dynamic programming on the bipartite graph ($R_{\text{-}2}$ is the corresponding production of the relation and $R_{\text{-}3}$ is the corresponding goal of the production).

$$H_{\mathcal{G}}(A \mid \mathcal{K}^E) = \begin{cases} \mathbb{I}_{A \in \mathcal{G}} & \text{if } Parent(A) = \varnothing \\ \sum_{R \in Parent(A)} u(R_{\text{-}2}) \cdot H_{\mathcal{G}}(R_{\text{-}3}) & \text{otherwise} \end{cases} \tag{3.40}$$

Hence, the process of finding the immediate action for one step reduces to an informed tree search. Where the state space is the objects in environment knowledge $\mathcal{V}^E$, the starting frontier is the objects with

no parents $Inst(S'_{t+1}) \cup \mathcal{G}$ (object directly instantiated from observation or pre-specified goals), the goal is to reach an object that is an atomic action and acceptable $A \in \mathbb{O}_A \wedge Acc(A \mid \mathcal{V}^E)$.

The overall action sequence returned towards the goal can be seen as the exploration process of a learning real-time A* search with one successor [41] where the heuristic update is done by changing the applicability of each production and meta productions given the new observations.

Production Reinforcement Learning

Following previous work in visual navigation [8], the agent has to explicitly choose the special `done` action to indicate that it has completed the current task. We further extend this and give the agent a `quit` option to indicate that it believes the given task is impossible in the given environment. This is important as we allow the architecture to choose to attend to any subtask as it wants, and it should be able to realize when a task is impossible.

We give a unit reward whenever the agent decides it is done with a task. Setting the rewards to be of different values to reflect different subtask priorities is unnecessary because that is reflected by the utility of the productions that propose these subtasks. The reward propagates back through the shortest path to the starting state. For example, if the state transition is

$$\mathcal{K}_0^E \xrightarrow{P_1} \mathcal{K}_1^E \xrightarrow{P_2} \mathcal{K}_2^E \xrightarrow{P_3} \mathcal{K}_0^E \xrightarrow{P_4} \mathcal{K}_4^E \xrightarrow{P_5} \mathcal{K}_5^E \xrightarrow{P_{\text{done}}}$$

where $\mathcal{K}_0^E$ is the start state and P_{done} is the production that yields the `done` action. Then the shortest path is

$$\mathcal{K}_0^E \xrightarrow{P_4} \mathcal{K}_4^E \xrightarrow{P_5} \mathcal{K}_5^E \xrightarrow{P_{\text{done}}}$$

Therefore only $P_4, P_5, P_{\text{done}}$ will receive a utility update, using the Bellman update [79].

$$u_{\text{after}}(P) \leftarrow \frac{1}{N(P)+1}\left(N(P) \cdot u_{\text{before}}(P) + \gamma^{\Delta_t}\right) \tag{3.41}$$

Where $u(P)$ is the utility of production P, $N(P)$ is the number of times P gets applied, Δ_t is the time difference from production application to the `done` action, and γ is the discount factor.

When a subtask is involved, the utility is updated with respect to each task. This is to ensure that the utility of production is stable during the course of training. The utility of the subtask production should not be penalized just because it is embedded in a longer-horizon task. For example, if the state transition is

$$A_0 \xrightarrow{P_1} A_1 \xrightarrow{P_2} \underbrace{B_3 \xrightarrow{Q_3} B_4 \xrightarrow{Q_4} B_5 \xrightarrow{Q_{\text{done}}}}_{\text{a subtask initiated by } P_2} A_6 \xrightarrow{P_{\text{done}}}$$

Where A and P correspond to the states and productions of the original task respectively and B and Q correspond to the states and productions of the subtask respectively. This will be treated as two separate utility update pathways

$$A_0 \xrightarrow{P_1} A_1 \xrightarrow{P_2} A_6 \xrightarrow{P_{\text{done}}} \qquad \text{and} \qquad B_3 \xrightarrow{Q_3} B_4 \xrightarrow{Q_4} B_5 \xrightarrow{Q_{\text{done}}}$$

If a subtask ends up with `quit` then there will be no utility update on its productions, not even negative ones. Because the task might be impossible due to environmental constraints, which has nothing to do with the production rules. Intuitively, the closer a production brings the agent to choose `done` for its current task, the higher its utility is.

Explainability

This framework touches upon all three aspects of explainability as defined in [55]. The preconditions of the productions directly specify the feature that is being used (feature importance). Each weight update corresponds to an exact trajectory which helps determine the training points that influence the learned policy the most (learning process). Lastly, the production application process can be easily converted to a verifiable decision tree by merging the precondition checks of productions (policy-level explainability).

3.3 Summary

In this chapter, we first formally define the building blocks of the agent framework - how attribute values are encoded as timelines, how production applications are tested, and how the world model is partially observable. Then using these definitions we explain how an agent is constructed, what its policy is, and how it can perform reinforcement learning based on experiences. Appendix A compares the proposed framework with other agent architectures.

Chapter 4

Bootstrapping World Knowledge

This chapter [1] illustrates how general knowledge of the world and the subgoal proposing productions can be extracted from large language models. Extending from previous work that generates plans for specific tasks [48, 75], defines the goal predicates for a task [37, 49], translates specific tasks into PDDL configurations [50, 88], or leverages existing learning capabilities of cognitive architecture [38, 70], our approach focuses on generating parameterized production rules that represent different chunks of *context-dependent knowledge*. Preliminary experiments show that it is hard to generate desirable production rules relying purely on prompt engineering. Therefore, we developed a bootstrapping framework (Figure 4.1) that generates abstract production rules from specifically grounded scenarios step-by-step and revises them through post-processing.

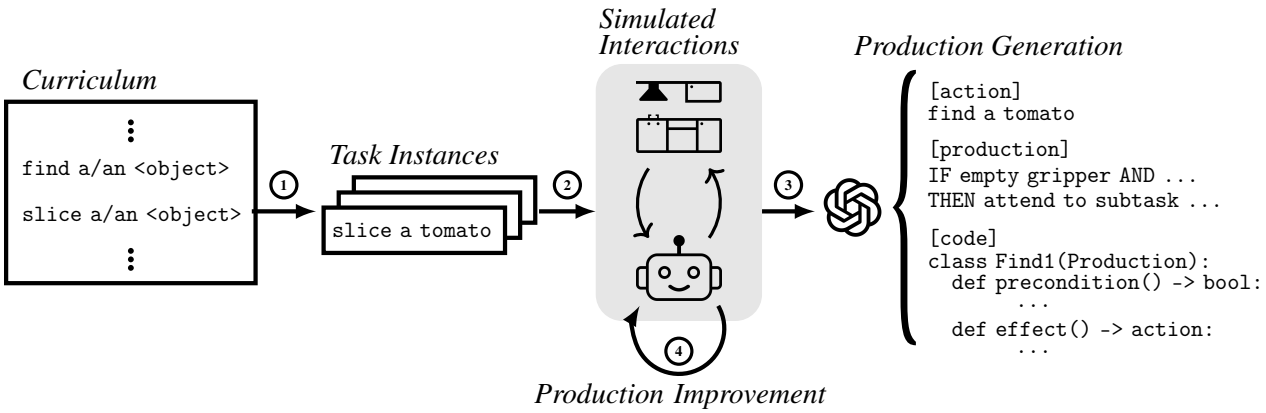


Figure 4.1: Overview of the bootstrapping process. It starts with a curriculum. For each family of tasks in the curriculum, a list of specific tasks is instatianated (1). Then the agent is set to interact with the environment attempting the specific task (2). When the agent does not know what to do, it queries the large language model to acquire a new production rule (3). And when the agent has accumulated sufficient knowledge, it does post-processing for production improvement (4).

[1] This chapter is based on [98], and uses a simplified version of the architecture described in the previous chapter where only one subtask is allowed for such (sub)task.

4.1 Curriculum and Task Instances

Following previous work on agent learning [85], we start with a *curriculum*, which is a list of task families. The advantage of using a curriculum is threefold: 1) it encourages a modular learning paradigm as the task learned first can be re-used for later tasks, 2) it ensures the consistency between the name of the tasks, and 3) it ensures task-decomposition respects the affordance of the robot as the agent is instructed to only use previously learned skills. This is similar to algebra education for human students where they start with simple arithmetics, then gradually move on to calculus where the later topics build on previously learned materials. Despite the same name, this is slightly different from "curriculum learning" in machine learning [12] as "curriculum learning" refers to training the model to do a single task with exemplars with increasing difficulty while the curriculum in our bootstrapping process consists of multiple families of tasks.

The most important feature of the curriculum in our framework is that it only needs the *the name of the task families*. No goal conditions or potential strategies are needed. The bootstrapping process will extract all those information from the large language model and this is the key to minimizing human effort. Concretely, we follow the SOAR syntax and keep all variables in angle brackets (e.g., `find a/an <object>` represents the family of finding tasks).

Analogous to how human students learn the principles of algebra with practice problems, specific task instances are used to bootstrap more generalized productions. We make extensive use of a simulator to randomly generate training tasks for the agent. For example, given the family of tasks in the pre-defined syntax (e.g., `find a/an <object>`), the environment randomly replaces the placeholder with a specific instance in the environment (e.g., `find a tomato`). Note that the task instance does not have to be achievable, as we want the agent to learn not only how to perform the task successfully but also how to identify when a task is impossible.

The agent might not fully learn every scenario of a task before moving on to the next one, it can still query the LLM later on to generate a production rule for a previously learned task. The training of a task is considered complete as long as the agent has sufficient experience with the task to generate a reasonable end condition (i.e., when the task is considered completed. In practice this is the summary of the precondition of all `done` productions for the task) such that future tasks can reuse the previously learned tasks.

4.2 Production Rule Generation

Action Prompting

The first step[2] of generating a production rule is to pick a specific action in a given concrete scenario. The LLM is prompted with the current task, a summary of the current state, and a list of options available to the robot, which include both motor actions on the environment (e.g., moving to a specific location) and internal actions (e.g., attending to a new subtask which is a task learned earlier in the curriculum). For each previously trained subtask, we provide the end condition generated by the critic for the LLM to evaluate its relevance (details described later in this section). Like the task names, the actions can also be parameterized (e.g., `move to <receptacle>`), and the LLM has replaced `<receptacle>` with a specific item as it sees fit.

[2]Examples of each step can be found in Appendix C.

We use chain-of-thought prompting [87], which explicitly instructs the LLM to respond to the prompt in a step-by-step manner, probing it to make the most informed decision. The LLM is instructed to reflect on common strategies for approaching the task, analyze the current situation, and evaluate the usefulness of each action before suggesting one option for the robot to take. The LLM is also prompted to state the purpose of the chosen action, which will inform the production rule generation later.

Production Prompting

Although the production rules are generated based on the current state, we represent them not as plans for the current task, but instead as underlying decision-making principles for all similar scenarios. For example, if the current task is to `find a/an egg`, instead of suggesting the action sequence of exploring every cabinet in the current environment, a desirable production rule would suggest "whenever you need to find something, you should first explore the unexplored places where that object is commonly stored". This is a systematic generalization that can be applied to finding any objects, not just eggs, and also can be applied to novel environments with different layouts and receptacle types.

To generate desired production rules, we use a two-step process. The first step summarizes the action selection process and generates the English description of the production rule; the second step then converts it into executable Python code.This separation is inspired by how human beginners are instructed to build cognitive models [43], and has two benefits: 1) it allows each query to the LLM to be of reasonable length ($\sim 5k$ tokens), preventing LLMs from losing focus on lengthy prompts [51]; and 2) it facilitates a modular design, which enables generating code from English descriptions generated from other sources, including human feedback and post-generation self-reflection.

For each step, we also use the chain-of-thought prompting technique. For English description generation, the LLM is given the entire history of the action selection process and is instructed to take four steps: 1) identify relevant information that leads to choosing the action; 2) generate a specific production rule that describes the current situation; 3) identify the potentially generalizable components in the specific rule and how they can be generalized; and 4) replace the components to form the generalized production description.

Code Generation

For code generation, the LLM is given the Python interface of querying declarative memory and the current task, and is instructed to take another four steps: 1) plan what variable bindings are needed; and how their values should be assigned, 2) analyze the predicates in the precondition and associate them with relevant variables; 3) plan how each predicate should be tested using the provided function interfaces; and 4) fill in the production template. The code snippet is parsed from the response and imported into the agent.

Similar to the iterative prompting design in Voyager [85], the agent replays the generated production rule on the state from which it was generated, and ensures that its precondition check passes the current conditions. This fixes most function interface mismatches, as the generated production has to comply with a specific naming scheme and the interface of the declarative knowledge.

Cycle Detection

Over-generalization happens when important features are left out of the production's precondition. For example, for the `pick and place` task, the LLM might generate a production rule that says:

```
IF task is pick and place <object> AND <object> in field of view AND gripper is empty
THEN pick <object>
```

This will make the robot pick up the object even when the object is already in the target receptacle. To prevent the agent from being stuck in an infinite loop, it will keep a state transition graph during the execution process and query the LLM for an alternative action once a cycle is detected using a depth-first search on the transition graph. Coupled with the production reinforcement (described in the previous chapter), the agent will prioritize loop-breaking productions.

End Condition Summarization

To re-use previously learned tasks as subroutines, we use a critic LLM to summarize the end condition of the learned tasks. The LLM is given all the production rules that condition on the selected task and is instructed to generate a one-sentence end condition based on all the supplied production rules whose effect is the 'done' action.

The end condition will be included in the future prompts such that when selecting an action in the future, the LLM knows what each subroutine is capable of. The advantage of this approach is twofold: 1) the agent can generate the end condition by itself without relying on human input, and 2) the end condition actually reflects what the agent *actually does* when attending to the subtask instead of what the human thinks the agent *should do*. For example, the bootstrapped agent defines an object being found as when the agent is holding it in its gripper instead of in its field of view.

Production Improvement

The end condition generated can also be used to guide batch post-processing on the production rules generated. This is important as the productions are individually generated so not necessarily constitute a coherent group.

As the LLM has access to accumulated observations from the past during the action selection process, it might include unnecessary conditions that happen to be true in the production's precondition, over-constraining it. For instance, the LLM might generate a production rule that says:

```
IF task is explore <receptacle>
AND <receptacle> is unexplored
AND NOT robot at <receptacle>
AND robot gripper is empty  # <-- redundant given robot affordance model
THEN move to <location>
```

This is handled by a critic LLM that provides suggestions on the existing productions through another step-by-step prompting.

The critic LLM will be given the name of the task, the summary of the end condition (as generated previously), and the list of all the English descriptions of the production rules related to the task. Then it is instructed to 1) list some common cases and intermediate steps of the task, 2) for each of the existing production rules, decide whether it should be kept as is, removed, or modified, 3) if any production rules need modification, generate the revised English description, 4) generate production rules for the missing cases. Using the code generation described previously, we can update the production rules or generate new production rules based on the revised English descriptions. An example is included in Appendix C.

4.3 Declarative Knowledge Integration

In addition to bootstrapping the production rules as the procedural knowledge of the agent, we also need to extract the knowledge about the world from the large language model. These can be used in the precondition functions of the production rules. For example, one of the production rule learned for `find` is

```
IF tasks is find <object>
AND NOT <object> in gripper
AND <object> is commonly stored in <location>  # <-- general world knowledge
AND NOT <location> is explored
THEN explore <location>
```

In this case the world knowledge is involved as part of the precondition test.

For the sake of simplicity, our implementation stores declarative knowledge in a dictionary. The dictionary maps sentences in natural languages to a Boolean value representing whether that statement is true or not. For instance

$$\texttt{Eggs are commonly stored in the fridge} \rightarrow \textit{True}$$
$$\texttt{Knifes are commonly stored in the fridge} \rightarrow \textit{False}$$

Note that unlike many early symbolic works that assume the absence of a predicate implies the negation of the predicate [26], we explicitly represent the truthfulness of a statement. An absence in the knowledge base implies that the agent has no knowledge on whether the statement is true or false.

This can be extended to use vector database [85] or existing ontologies [4] for better performance in a specific domain. But in this week we keep it simple and show that large language models are also sufficient for these type of world knowledge.

4.4 Experiment

Setup

Following previous works in the embodied agents domain [71, 83], we evaluate our method in kitchen environments in the AI2THOR simulator [39]. As shown in Figure 4.2b, the agent has access to classification labels and attributes (e.g., "is opened") for objects that are close enough (within $1.5m$) or large enough (more than 5% of the frame). We also assume the agent already knows the names and locations of the large receptacles (e.g., cabinets, fridges, etc.) but does not know what objects are in the receptacles until it actively explores them.

We use three different tasks for evaluation:

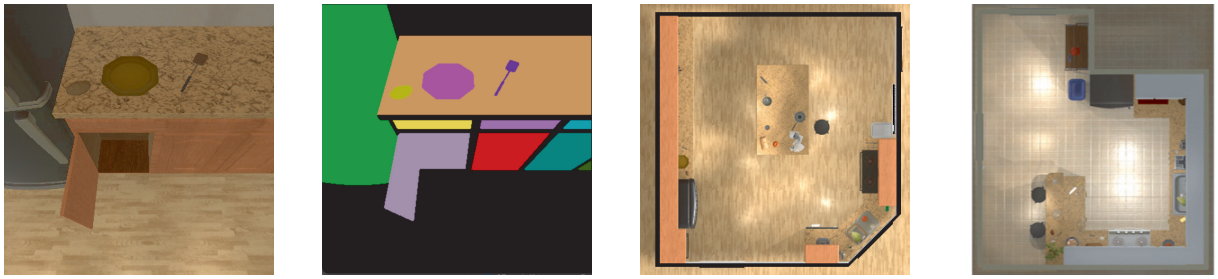
- `find a/an <object>`: the goal is to have the specified object in the robot's field of view. This is a fundamental skill that is often directly assumed in many of the previous works [75]. We want to show that our framework can bootstrap very basic skills, in addition to composite actions.
- `slice a/an <object>`: the goal is to use a knife to slice an object. Because the robot can hold at most one item at a time, slicing involves a sequence of actions including finding the target object and the knife, putting them in the same place, and the final slice action. We want to show that our framework can handle tasks that involve multiple steps and tool use.

- `clear the countertops`: the goal is to have all the objects on the countertops moved to suitable storage places. This is a common household task that has also been investigated in previous work [9, 71]. We want to show that our framework can handle tasks that involve repeating similar subtasks.

The goal conditions listed above are used only for evaluation purposes, but are not provided to the LLM during training or testing. The LLM has to infer the goal condition from the task description only.

For `find` and `slice`, 5 target objects are chosen for each task, and we run 3 trials for each object where the initial locations of the objects are shuffled. For `clear the countertops` we run 3 trials each with 5 objects on the countertops that need to be put away. The specific objects and locations vary between trials, and the success of the agent is evaluated based on how many objects originally on the countertops have been relocated to other places. This results in 15 specific goal instances for each task family.

We use `GPT4-0613` [60] for our experiments as previous works have shown that GPT3.5 is insufficient for code generation [59, 85]. We set the temperature to 0 for the most deterministic response.



(a) ego-centric view (b) object segmentation (c) training floor plan (d) testing floor plan

Figure 4.2: Screenshots of the AI2THOR simulator

Bootstrapped Agent

For the experiment condition, we bootstrapped our agent with the following curriculum in the training floor plan:

1. `explore <receptacle>`
2. `find a/an <object>`
3. `pick and place a/an <object> in/on a/an <receptacle>`
4. `slice a/an <object>`
5. `put things on the countertop away`

For each task family, random instances are chosen from the environment to instantiate 10 specific tasks. This process generated 27 production rules in total. Empirically it is shown that 10 instances is sufficient for learning the policy for each new task family.

During test time, the agent can query the LLM for an immediate action if it does not have an applicable production rule for the current situation, but it cannot learn new production rules. In the result section we report both with or without access to LLM during test time.

Baseline Condition

For the baseline condition of using LLMs to query only the actions, we omit the production generation steps and only use the action selection process within our framework. This ensures the prompts used by both conditions are the same, so LLM should suggest actions of similar quality. If the action proposed by

Task	Agent	Success (w/o LLM) ↑	Steps ↓	Tokens ↓
`find a/an <object>`	action-only	14(−)/15	15.67	54754.20
	bootstrapped no RL (ours)	15(12)/15	20.67	901.93
	bootstrapped (ours)	15(12)/15	15.80	916.87
`slice a/an <object>`	action-only	15(−)/15	28.20	102806.60
	bootstrapped no RL (ours)	15(15)/15	33.27	0.00
	bootstrapped (ours)	15(15)/15	29.13	0.00
`clear the countertops`	action-only	15(−)/15	5.13	18924.87
	bootstrapped no RL (ours)	15(15)/15	7.53	0.00
	bootstrapped (ours)	15(15)/15	7.47	0.00

Table 4.1: Results of experiments on household tasks. Completion steps and tokens are averaged over all task instances. The success rate was measured both with the assumption that the agent still has access to LLM during test time (numbers on the left) and without access (numbers in the parenthesis).

the LLM leads to an affordance error, we query LLM another two times, and if none of the actions are viable by the agent, then it is considered to have failed the task.

Quantitative Results

Table 4.1 shows the quantitative results of different types of agents performing each kitchen task. The action-only baseline successfully completes all tasks but one, where it assumes `find a/an mug` is equivalent to `find a/an cup` and ends the search prematurely without exploring the sink where the mug is actually located. On the other hand, our bootstrapped agent is able to finish most tasks completely using its learned production rules. The only exceptions are when it is tasked to find an object that was not part of its training environment. But with very limited additional queries, the bootstrapped agent is able to successfully complete those tasks as well. This shows that the knowledge in the bootstrapped agent can be easily transferred to new objects in new environments.

The success rate and number of query tokens show two advantages of our framework. First, because the precondition function is deterministic, there won't be unexpected false assumptions (e.g., a mug is the same as a cup). Second, it is much more efficient to be deployed into new environments as the production rules it learns can be easily transferred and require minimal further assistance from the LLM, saving computations and costs.

We use a paired sample t-test with Dunnett correction for multiple hypothesis testing to compare the number of steps taken by both agents. No significant evidence suggests that the two agents perform differently in `find` or `slice` tasks (p-values 0.446 and 0.347, respectively). This is not surprising as the knowledge source of both agents is the same LLM.

However, the number of steps taken for the baseline agent to complete the `clear` task is less than the bootstrapped agent with significance (p-value less than 0.001). This is analyzed in the next section.

Another notable difference is between using and not using reinforcement learning. We found that there is a statistically significant difference in the number of steps taken in the `find` (p-value 0.025) and

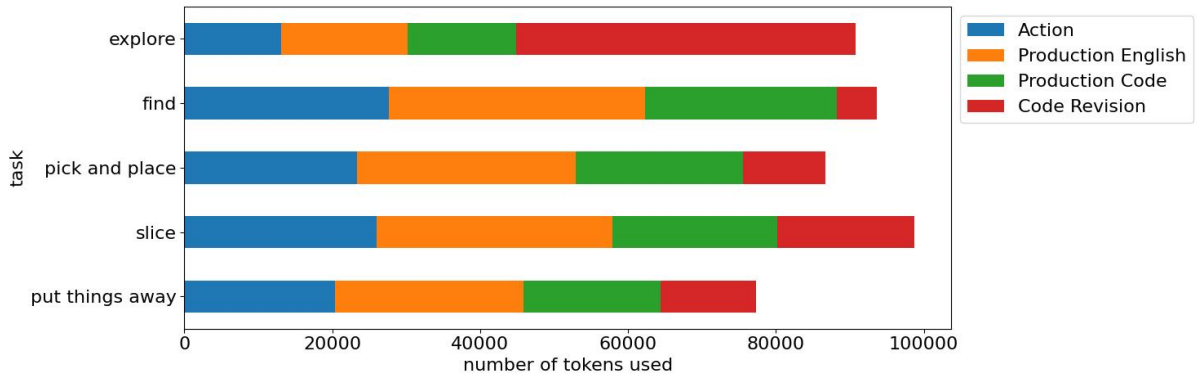


Figure 4.3: Token usage statistics during the bootstrapping process

`slice` (p-value 0.004) tasks between the baseline and no reinforcement learning on the production rules. This shows that learning the utility of the production rules is important for the efficiency of the cognitive agent. Without sampling based on the learned utility, the agent is more likely to execute redundant actions before choosing the `done` action.

Figure 4.3 shows that the number of tokens needed to train each task is roughly the same. So as the curriculum expands, the number of tokens needed will only grow linearly. Additionally, the number of tokens needed to train one task is less than one single trial of slicing objects of the action-only agent as reflected in Table 4.1. This shows that our framework is much more cost-effective. At the time of the experiment, the testing experiments on the baseline action-only agent cost around $120 in total while the bootstrapping of our framework costs less than $40.

Qualitative Analysis

The following are some learned productions:

- IF the current task is to find a/an `<object>` AND the `<object>` is located on `<location>` AND the robot is not at `<location>` THEN choose motor action: move to `<location>`.
- IF the current task is to slice a/an `<sliceable>` AND the robot is holding a/an `<sliceable>` AND there is no `<tool>` in the spatial knowledge or object knowledge THEN choose 'attend to subtask: find a/an `<tool>`'.
- IF the current task is to clear objects from a/an `<receptacle_type>` AND all the `<receptacle_type>` are empty THEN choose special action: 'done'.

These show that the agent is able to represent different aspects (atomic step, task decomposition, task end condition) of the given tasks using production rules. The first represents a common strategy for finding things, namely how to find things with a known location. The second represents decomposing complex tasks and reusing previously learned tasks. The third is a correct termination condition, which is not directly provided, for the exploration task from the LLM.

Figure 4.4 shows the dependencies between different tasks after the bootstrapping process. It is shown that the agent learns to break down more complex tasks into easier tasks as desired. It works for both sequential decomposition (the `slice` action involves a sequence of finding the knife, the object, navigating, etc.) and repetitive decomposition (the `clear` action involves doing the same pick and place procedure for all the objects). Note that despite there is no explicit loop structures in the agent architecture, loops

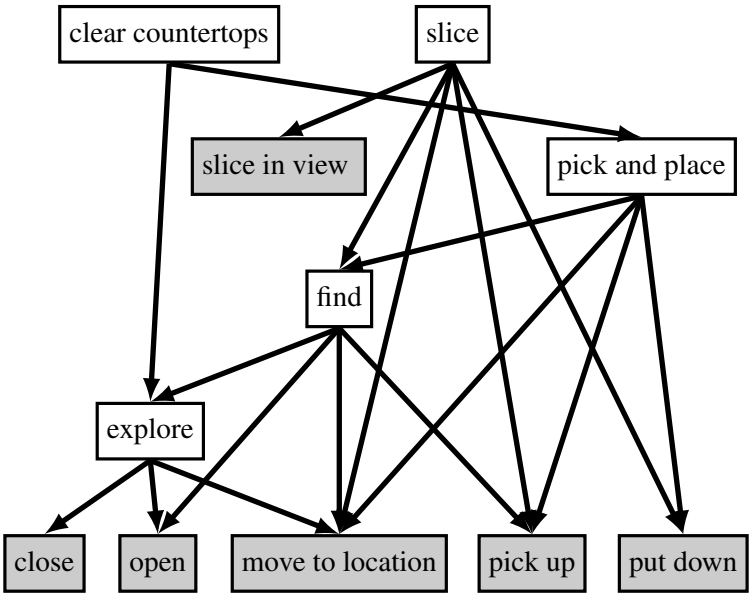


Figure 4.4: The hierarchy of tasks learned. Gray nodes denote the built-in functions of the robot, and white nodes represent the tasks learned from the curriculum. Arrows represent performing one task (tail) depending on using the other (head) as a subroutine. For built-in actions that involve an object (e.g., close), the object has to be within the field of view for the action to be taken. Special actions (i.e., done and quit) are omitted in the interest of space.

can be represented as recursive applications of production rules. This complements the statistics in Figure 4.3 and together they demonstrate the scalability of our approach.

Additionally, we also noticed some stylistic differences between the bootstrapped agent and the baseline agent, which is essentially why the bootstrapped agent is taking longer in the `clear` task. As shown in Figures 4.5a and 4.5b, the bootstrapped agent will put each individual item in its own cabinet while the LLM-only baseline will put everything in the same cabinet. This is because the baseline makes individual decisions at each time step and optimizes towards the total number of steps. In contrast, the bootstrapped agent will re-use the knowledge it learned as much as possible - when it learned to put one item in its own cabinet, it will do the same for the other items. Specifically, the production rule it has is "if there is an object on the countertop and there is an empty receptacle, attend to the subtask pick up the object, and place it into the empty receptacle".

This is more apparent for the `slice` task where the agent would place the object to be sliced on the countertop before slicing it (Figure 4.5c) as it learns the proper action sequence of slicing, while the baseline agent will just slice the object wherever it is currently (Figure 4.5d).

4.5 Summary

This chapter describes the bootstrapping process of generating a cognitive agent using a large language model. It addresses the challenge of production rules being too tedious to generate by human annotators. The experiments have shown that the bootstrapped agent can perform tasks in the same amount of steps as a large language model alone but requires much less computation in terms of tokens generated as the production rules are reused in novel environments. Additionally, the production rules are also scalable

(a) ours on `clear` (b) baseline on `clear` (c) ours on `slice` (d) baseline on `slice`

Figure 4.5: Stylistic differences between the bootstrapped agent and the baseline

to more complex and long-horizon tasks because the agent learned to reuse previously learned tasks as subtasks.

Chapter 5

Personalized Adaptation

This chapter illustrates a pilot study on how the existing production rules in the agent can be adapted with respect to different user preferences.

5.1 Preference Collection

Linear Kitchen Environment

Despite the existence of many household simulators [1, 9, 39, 65], to the best of our knowledge none are very suitable for our purpose of collecting preferences, mainly because they lack object diversity in a specific environment (e.g., kitchen) or do not support object state changes (e.g., slice an object). Thus, to enable the users to provide more fine-grained preferences, we developed a toy kitchen environment named LinearKitchen[1]. It sacrifices more fine-grained object shapes and physics for the benefit of having more diverse objects.

Many of the previous environments are based on physics engines such as Unity [29] or Mujoco [82] because they are originally designed to simulate accurate manipulation or locomotion. This makes it hard to add new objects to the existing environment as defining a new object requires defining a 3D mesh and its appearances. As we assume access to ground truth perception and do not need an accurate physics model, we implemented a very naive rendering pipeline. This makes adding a new object very simple - all it needs is some screenshots from different views.

Another important design choice is to have object-centric representations as opposed to having a centralized data structure that keeps track of all the objects. This is to ease the process of rendering. The rendering process is reduced to a DFS on object appearances since the position of objects is represented by their support and the receptacles are at the root level.

Figure 5.1 shows the full rendering of the entire kitchen, with the robot resembling stretch [35].

Pilot User Study

Although many of the previous works discussed using their method for preference adaptation, the "preferences" they used are generated from templates and have human annotators to describe them in their own words instead of querying human users for their actual preferences [72, 73]. In other words, the

[1] As the name implies it is a 2D kitchen with all the receptacles lining up in a line.

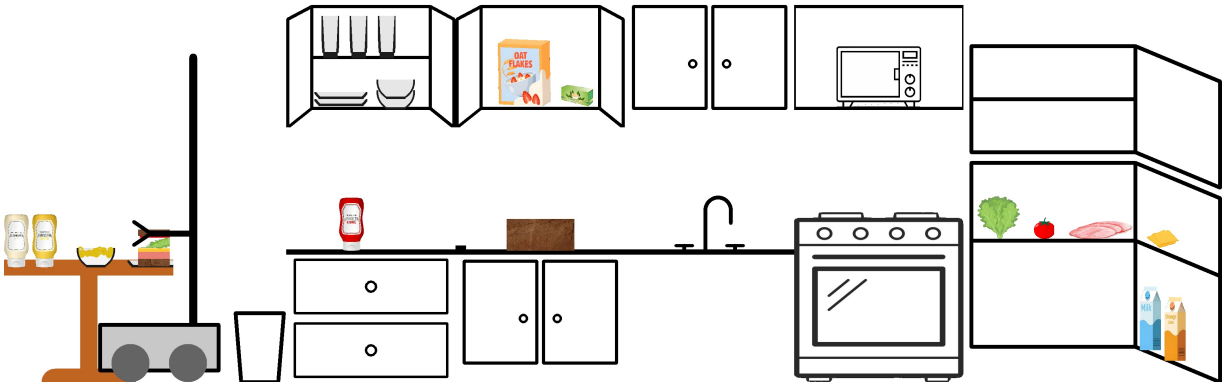


Figure 5.1: Complete view of the linear kitchen environment

experimenters control the set of preferences and the variance is in the language used to describe those predefined preferences. In this work, we want to collect both the diverse language used by the participants and more importantly the actual personal preferences in the wild.

Preliminary pilot studies and other research [25] have shown that people are not good at introspecting their preferences when the question is completely open-ended. Therefore we start by showing the annotators a video of the robot making breakfast (defined by a set of 28 hand-coded production rules to make a sandwich and a bowl of cereal) and then asking them what they would like the robot to have done differently to reflect their preference.

We collected a total of 46 utterances from 7 participants (3 male, 4 female, age 21-24) during the pilot study. We adopted an in-person interview setting where each participant is shown the sequence of the robot making breakfast and is instructed to reflect on what they would like the robot to change. The participant can rewind the video freely after the first pass. As shown in Figure 5.2, different participants care about different aspects of the breakfast - some want other items for breakfast, some are okay with sandwiches but want it to be done differently, some have spatial preferences on where things are located, etc. This verified that people have diverse preferences.

5.2 Production Rule Modification

Given the modular approach in the previous chapter for generating new production rules, we can assume that given a production description in the form of `IF ... THEN ...`, GPT-4 is capable of translating it into code. Therefore, the production rule modification only needs to happen in the language domain, which is desirable since currently large language models like GPT-4 are mainly trained using natural language datasets. In this work, we also update the preferences one utterance at a time to make it easy for the large language models.

To do so we again use the chain-of-thought prompting technique where we instruct the model (`GPT4-0613`, same model as in the previous chapter with temperature 0) to first categorize the type of the preference into one of the given predefined categories (Breakfast Item, Variation, etc.), then plan how the preference can be expressed using proposing production rule, and finally go through the list of existing production rules and make edits to the relevant ones. During the editing process, the large language model can decide if each of the current production should be kept as is, removed completely, or modified to have new preconditions or effects. The language model is also instructed to "give up" on the preference that it believes cannot be expressed using production rules.

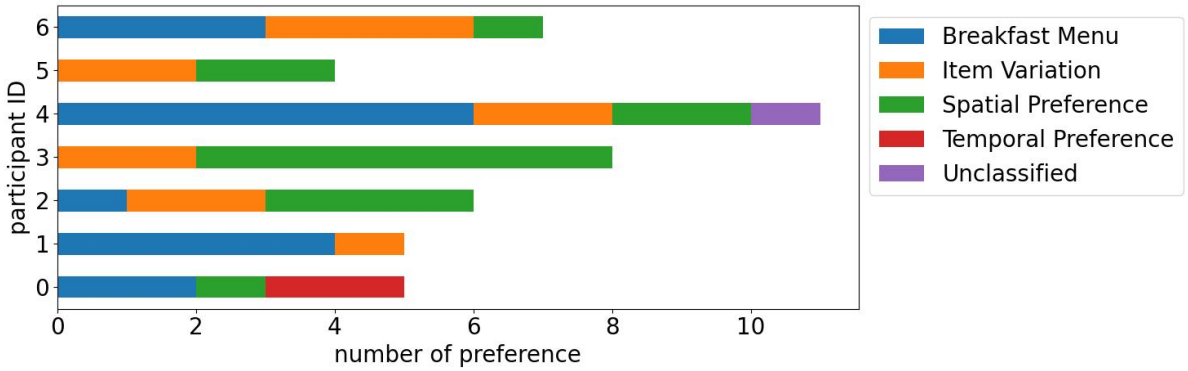


Figure 5.2: Breakdown of participant preference category. “Menu Item" refers to wanting other items for breakfast that are not sandwiches and cereal (e.g., “I want fruits in my breakfast"). “Item Variation" refers to having their sandwich or cereal be prepared differently (e.g., I want less milk in my cereal). “Spatial Preference" refers to having objects placed in specific locations (e.g., put back the ketchup bottle). “Temporal Preference" refers to having a specific order of the breakfast (e.g., cutting two slices of bread before assembling it). The only “Unclassified" preference is “don’t burn the kitchen". Each utterance is assigned a unique category.

The overall production modification procedure is relatively simple compared to the bootstrapping process described in the previous chapter. This is intentional as one of the core ideas of the architecture is to make it easy to update to fit different preferences.

5.3 Results

Evaluation Criteria

All the production rules generated by GPT were manually inspected for correctness and we adopted a very lenient evaluation criteria. When the preference is ambiguous, it is considered correct as long as the updated production meets the preference. For example, when the preference is “I want a smaller breakfast", we consider the updated productions to be correct if they make an open-face sandwich instead of the original sandwich (the new sandwich is one piece of bread smaller than the original).

Additionally, we assume a new task can be learned from the bootstrapping process as described in the previous chapter. Therefore, when the user wants a new item for their breakfast (e.g., a pancake), the update productions are considered correct if they successfully update 1) the subtask proposing (e.g., propose making a pancake when the task is to make breakfast) and 2) the end condition (e.g., making a breakfast is considered done if there is a pancake). It is irrelevant whether the new productions can actually make a pancake.

Quantitative Analysis

Figure 5.3 shows the success rate of preference adaptation grouped by preference category. Overall 38 out of 46 (82.6%) utterances were successfully expressed by the updated productions. The chi-square test

Failure Type	Misunderstanding	Incorrect Precondition	Irrelevant Edit	No Attempt	Affordance Error
# Occurrences	3	2	1	1	1

Table 5.1: Failure cases for preference adaptation

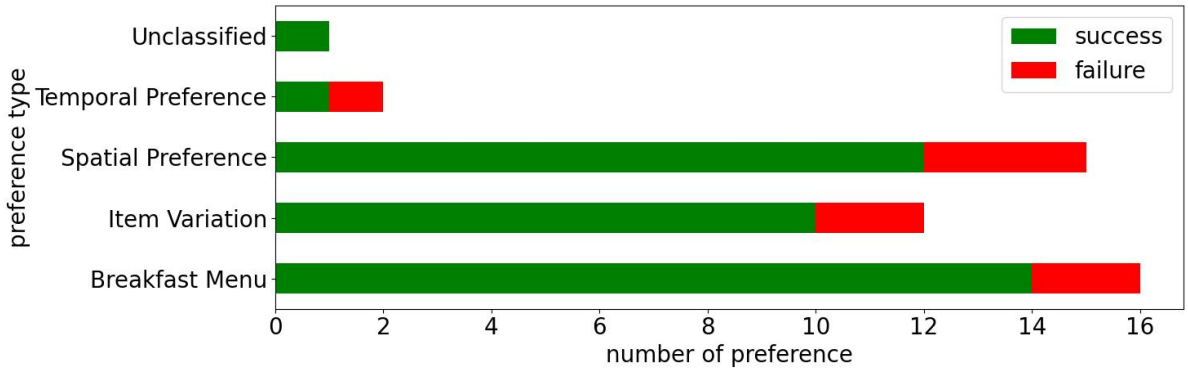


Figure 5.3: Preference adaptation success rate by category

shows that there is no evidence suggesting that the agent can adapt to one category of preference better than others (p-value 0.618).

Qualitative Analysis - Failure Cases

Table 5.1 categorizes the type of failures made by the model. The most common category is misunderstanding the user's preference expression. For example, one of the users said "bread should be in cabinets" and GPT4 treated it as a precondition that directs the robot to look into cabinets for bread instead of putting the bread into the cabinet. The other two misunderstanding cases are where the user says "I want fruit, I want berries" but the model fails to realize berries are a type of fruit and proposes to make a fruit bowl and have another bowl of berries. These failures are due to the natural language processing capabilities of the language model.

Another common mode of failure is when the modifications are on the right track but have some minor issues with the productions. For example, one of the incorrect production rules generated is

```
IF the task is to heat up milk AND the milk is not hot
THEN DONE heat up milk
```

This might originate from the hallucination issue of language models. This also happens to the irrelevant edit case where the model updates the preference of cereal-making when the user wants something warm.

The large language model also gave up on one of the preferences of "cut two pieces of bread in one go" by making no attempt, stating that it cannot be expressed using the production rules. This showed that with few-shot prompting the large language model cannot conjure more complex production rules to represent strategies that are not already in the agent. And lastly, there is an affordance error case where the model incorrectly assumes the robot can directly melt the cheese on the sandwich without putting it in an oven.

Qualitative Analysis - Success Cases

Most edits to the breakfast menu are optimal, which is likely attributed to having direct examples in the existing policy on how to propose new subtasks and update the end conditions of making a breakfast. The model is also very good at making reasonable edits to meet the user's needs. Examples include replacing the ham slices with tomato slices for vegetarians or using the toaster to toast the bread. Also, GPT4 gave up on dealing with "don't burn down the kitchen", which seems like a reasonable response.

However, GPT4 is not currently good at making use of comparison productions. Instead of specifying the user's preference as comparisons, it just codes the preference directly into the production rules. This is likely caused by having only one comparison production example in the prompt. Similarly, when dealing with vegetarian preference, GPT4 will replace the ham slice with a tomato slice but not explicitly encode the vegetarian preference. It technically produces the behavior that fits the user's preference but is less ideal as it leads to production rules with poor modularity and it would be hard to transfer the learned production rule to other similar users (e.g., vegetarian users that don't like tomatoes.)

5.4 Summary

This chapter describes the pilot user study we ran to demonstrate how the architecture can adapt to the user's needs in one shot. We constructed a new environment to support more detailed breakfast preferences. Then using a video clip generated in the environment, we gathered user preferences of different types. We showed that with a simple prompting pipeline in the language space, the agent can successfully accommodate more than 80% of the preference requirements.

Chapter 6

Discussion

6.1 Limitations

Although we allow the precondition function of each production to be more than symbol testing, currently it is restricted to give a binary output on the applicability of the production. This relies heavily on having ground truth perceptions and does not take the uncertainty into account.

Additionally, the current bootstrapping process and preference adaptation require procedural knowledge to be articulable as the language space bridges the human preference and the code executed by the agent. This might be hard for motor skills such as sculpting.

The preference adaptation is still in a very primitive stage where there is only a single round of interaction and the preference is being interpreted very loosely. Additionally, the current adaptation process goes through all the existing production rules in the agent which could be computationally expensive. Moreover, the preferences are collected from a specific population (i.e., college students) which might have limited external validity.

6.2 Future Work

Integrated System

Previous chapters described the essential components of such a framework - the architecture, how the production rules can be bootstrapped, and how an existing set of production rules can be adapted w.r.t human input. However, in this work, we didn't put everything together to form an integrated system yet. So here we provide a brief sketch of what other components are needed to form a holistic adaptable agent.

Perception-wise one can make use of existing object-centric segmentation models [74] coupled with ongoing work in open-vocabulary attribute detection models [16] to acquire the structuralized perception input assumed in the framework. In terms of execution, there have been many works in low-cost robot teleoperation pipelines for behavior cloning [21, 33]. These can be used to define the action primitives used in our framework when deploying the architecture on physical robots. And finally, existing speech recognition tools [68] can be used for taking users' feedback as input.

We believe all the components needed for a physically interacting robot have been developed by the research community. With some engineering efforts, an integrated system that interacts and adapts to a user's use case should be feasible.

User Acceptance Study

In this work, we focused primarily on the technological part of the architecture but didn't explore how other people would perceive such an agent. The hypothesis is that agent running production rules in a cognitive architecture would be more interpretable and thus more trustworthy than large foundation models by themselves. But to test the hypothesis would require more long-term user study.

Additionally, user acceptance also depends on many other factors. Potential complaints could be the system is not flexible enough or it is too much burden to correct the agent's behavior once in a while. Or simply a layperson might not perceive it to be as astonishing as a large foundation model [69].

Neural Production Rules

In the long run, we believe having neural production rules that operate in the continuous domain is necessary for the agent to adapt to the edge cases in the real world and eventually incorporate motor skills. Despite some existing work exploring neural production rules [10] and integrating neural sampler into symbolic planners [54], the existing works stay in toy domains (e.g., 2D grid world). And it remains an open question of how to build a lifelong learning agent that can pick up a wide range of skills.

Here we introduce a very rough idea for the future direction. First train multiple generic feature functions (e.g., color, sharpness etc.) using diffusion models [22] or other weakly supervised models. Then we can represent objects as indexable production rules (i.e., each production rule will be associated with an embedding vector), where the precondition for atomic objects is a set of classifiers based on the previously trained diffusion models [47] and the composite objects will have other atomic objects as preconditions. Next, we should also train the embedding space of the production such that semantically similar objects have production embedding close to each other and ideally have the space aligned with the language space using CLIP style training [67]. This ensures that when building up new concepts (objects or policies), we can easily retrieve relevant prior knowledge and reuse them.

Chapter 7

Conclusion

In this work, we developed a framework for preference adaptation. Unlike the prevalent approaches that use rewards as proxies, we were inspired by previous work in cognitive architecture to directly encode preferences as production rules that define the agent policy based on the context. It takes advantage of the abstraction, modularity, and composability aspects of production rules. We formally defined the agent's policy based on each component, explained a bootstrapping process so that the agent can acquire basic skills from large language models with minimal human supervision, and finally demonstrated that it can be used to adapt to more than 80% of the breakfast preferences collected in real life. In the end, we also discussed the limitations of the current approach and how that informs future work in this direction.

We believe this work shows that it is possible to encode most human preferences as rules and perform adaptation easily. We hope it will inspire more future work in rule-centric models from preference adaptation and more general human-AI interaction.

Bibliography

[1] Robohive – a unified framework for robot learning. https://sites.google.com/view/robohive, 2020. URL https://sites.google.com/view/robohive. 5.1

[2] Charu C Aggarwal et al. *Recommender systems*, volume 1. Springer, 2016. 2.1

[3] 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. 2.3, **??**

[4] Florian Ahrens, Mihai Pomarlan, Daniel Beßler, Thorsten Fehr, Michael Beetz, and Manfred Herrmann. Towards a neuronally consistent ontology for robotic agents. *arXiv preprint arXiv:2309.14841*, 2023. 4.3

[5] Erik M Altmann and J Gregory Trafton. Timecourse of recovery from task interruption: Data and a model. *Psychonomic Bulletin & Review*, 14(6):1079–1084, 2007. 2.2

[6] John R Anderson, Daniel Bothell, Michael D Byrne, Scott Douglass, Christian Lebiere, and Yulin Qin. An integrated theory of the mind. *Psychological review*, 111(4):1036, 2004. 2.2, 3, 3.2, **??**

[7] John R Anderson, Shawn Betts, Daniel Bothell, and Christian Lebiere. Discovering skill. *Cognitive Psychology*, 129:101410, 2021. 2.2

[8] Peter Anderson, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, et al. On evaluation of embodied navigation agents. *arXiv preprint arXiv:1807.06757*, 2018. 3.2

[9] Szot Andrew, Yadav Karmesh, Clegg Alex, Berges Vincent-Pierre, Gokaslan Aaron, Chang Angel, Savva Manolis, Kira Zsolt, and Batra Dhruv. Habitat rearrangement challenge 2022. https://aihabitat.org/challenge/rearrange_2022, 2022. 4.4, 5.1

[10] Goyal Anirudh, Aniket Didolkar, Nan Rosemary Ke, Charles Blundell, Philippe Beaudoin, Nicolas Heess, Michael C Mozer, and Yoshua Bengio. Neural production systems. *Advances in Neural Information Processing Systems*, 34:25673–25687, 2021. 6.2

[11] Nolan Bard, Jakob N Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, et al. The hanabi challenge: A new frontier for ai research. *Artificial Intelligence*, 280:103216, 2020. 2.1

[12] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009. 4.1

[13] Erdem Biyik and Dorsa Sadigh. Batch active preference-based learning of reward functions. In *Conference on robot learning*, pages 519–528. PMLR, 2018. 2.1

[14] Dan Bohus, Sean Andrist, Ashley Feniello, Nick Saw, and Eric Horvitz. Continual learning about

objects in the wild: An interactive approach. In *Proceedings of the 2022 International Conference on Multimodal Interaction*, pages 476–486, 2022. 2.1

[15] Daniil A Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023. 2.3

[16] Maria A Bravo, Sudhanshu Mittal, Simon Ging, and Thomas Brox. Open-vocabulary attribute detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7041–7050, 2023. 6.2

[17] John S Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. *arXiv preprint arXiv:1301.7363*, 2013. 2.1

[18] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023. **??**

[19] Erdem Bıyık, Malayandi Palan, Nicholas C. Landolfi, Dylan P. Losey, and Dorsa Sadigh. Asking easy questions: A user-friendly approach to active reward learning, 2019. 2.1

[20] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023. 1

[21] Cheng Chi, Zhenjia Xu, Chuer Pan, Eric Cousineau, Benjamin Burchfiel, Siyuan Feng, Russ Tedrake, and Shuran Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. *arXiv preprint arXiv:2402.10329*, 2024. 6.2

[22] Yilun Du and Igor Mordatch. Implicit generation and modeling with energy based models. *Advances in Neural Information Processing Systems*, 32, 2019. 6.2

[23] Tesca Fitzgerald, Pallavi Koppol, Patrick Callaghan, Russell Quinlan Jun Hei Wong, Reid Simmons, Oliver Kroemer, and Henny Admoni. Inquire: Interactive querying for user-aware informative reasoning. In *6th Annual Conference on Robot Learning*. 2.1

[24] Riccardo Fogliato, Shreya Chappidi, Matthew Lungren, Paul Fisher, Diane Wilson, Michael Fitzke, Mark Parkinson, Eric Horvitz, Kori Inkpen, and Besmira Nushi. Who goes first? influences of human-ai workflow on decision making in clinical imaging. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, pages 1362–1374, 2022. 2.1

[25] Craig Fowler, Jian Jiao, and Margaret Pitts. Frustration and ennui among amazon mturk workers. *Behavior Research Methods*, 55(6):3009–3025, 2023. 5.1

[26] Maria Fox and Derek Long. Pddl2. 1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003. 3.1, 4.3

[27] Wai-Tat Fu and John R Anderson. Extending the computational abilities of the procedural learning mechanism in act-r. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, volume 26, 2004. 2.2

[28] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023. 2.3

[29] John K Haas. A history of the unity game engine. 2014. 5.1

[30] Holly Sue Hake, Catherine Sibert, and Andrea Stocco. Inferring a cognitive architecture from multi-

task neuroimaging data: A data-driven test of the common model of cognition using granger causality. *Topics in Cognitive Science*, 14(4):845–859, 2022. 2.2

[31] Donald Joseph Hejna III and Dorsa Sadigh. Few-shot preference learning for human-in-the-loop rl. In *Conference on Robot Learning*, pages 2014–2025. PMLR, 2023. 1

[32] Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*, 2023. 2.3

[33] Aadhithya Iyer, Zhuoran Peng, Yinlong Dai, Irmak Guzey, Siddhant Haldar, Soumith Chintala, and Lerrel Pinto. Open teach: A versatile teleoperation system for robotic manipulation. *arXiv preprint arXiv:2403.07870*, 2024. 6.2

[34] Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. Vima: General robot manipulation with multimodal prompts. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022. 2.3

[35] Charles C Kemp, Aaron Edsinger, Henry M Clever, and Blaine Matulevich. The design of stretch: A compact, lightweight mobile manipulator for indoor human environments. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 3150–3157. IEEE, 2022. 5.1

[36] 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. Segment anything, 2023. 2.3, 3.1

[37] James R. Kirk, Robert E. Wray, Peter Lindes, and John E. Laird. Integrating diverse knowledge sources for online one-shot learning of novel tasks, 2023. 2.2, 4

[38] James R. Kirk, Robert E. Wray, Peter Lindes, and John E. Laird. Improving knowledge extraction from llms for task learning through agent analysis, 2024. 2.2, 4, **??**

[39] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi. AI2-THOR: An Interactive 3D Environment for Visual AI. *arXiv*, 2017. 4.4, 5.1

[40] Pallavi Koppol, Henny Admoni, and Reid Simmons. Iterative interactive reward learning. 2.1

[41] Richard E Korf. Real-time heuristic search. *Artificial intelligence*, 42(2-3):189–211, 1990. 3.2

[42] Iuliia Kotseruba and John K Tsotsos. 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artificial Intelligence Review*, 53(1):17–94, 2020. 2.2

[43] John E Laird. Soar 9.6.0 tutorial, Jun 2017. URL https://soar.eecs.umich.edu/articles/downloads/soar-suite/228-soar-tutorial-9-6-0. 4.2

[44] John E Laird. Introduction to soar. *arXiv preprint arXiv:2205.03854*, 2022. 2.2, 3, 3.2

[45] John E Laird, Kevin Gluck, John Anderson, Kenneth D Forbus, Odest Chadwicke Jenkins, Christian Lebiere, Dario Salvucci, Matthias Scheutz, Andrea Thomaz, Greg Trafton, et al. Interactive task learning. *IEEE Intelligent Systems*, 32(4):6–21, 2017. 2.2

[46] John E Laird, Christian Lebiere, and Paul S Rosenbloom. A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. *Ai Magazine*, 38(4):13–26, 2017. 2.2

[47] Alexander C Li, Mihir Prabhudesai, Shivam Duggal, Ellis Brown, and Deepak Pathak. Your diffusion

model is secretly a zero-shot classifier. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2206–2217, 2023. 6.2

[48] 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. 4

[49] James R Lindes and Wray Peter. Improving knowledge extraction from llms for robotic task learning through agent analysis. *arXiv preprint arXiv:2306.06770*, 2023. 4

[50] Bo Liu, Yuqian Jiang, Xiaohan Zhang, Qiang Liu, Shiqi Zhang, Joydeep Biswas, and Peter Stone. Llm+ p: Empowering large language models with optimal planning proficiency. *arXiv preprint arXiv:2304.11477*, 2023. 4

[51] Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv preprint arXiv:2307.03172*, 2023. 4.2

[52] Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*, 2023. 2.2

[53] Jiayuan Mao, Tomás Lozano-Pérez, Josh Tenenbaum, and Leslie Kaelbling. Pdsketch: Integrated domain programming, learning, and planning. *Advances in Neural Information Processing Systems*, 35:36972–36984, 2022. 3.1

[54] Jorge Mendez-Mendez, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Embodied lifelong learning for task and motion planning. In *Conference on Robot Learning*, pages 2134–2150. PMLR, 2023. 3.1, 6.2

[55] Stephanie Milani, Nicholay Topin, Manuela Veloso, and Fei Fang. A survey of explainable reinforcement learning. *arXiv preprint arXiv:2202.08434*, 2022. 3.2

[56] Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A Feder Cooper, Daphne Ippolito, Christopher A Choquette-Choo, Eric Wallace, Florian Tramèr, and Katherine Lee. Scalable extraction of training data from (production) language models. *arXiv preprint arXiv:2311.17035*, 2023. 2.3

[57] Allen Newell. You can't play 20 questions with nature and win: Projective comments on the papers of this symposium. 1973. 2.2

[58] Allen Newell. *Unified theories of cognition*. Harvard University Press, 1994. 2.2

[59] Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. Demystifying gpt self-repair for code generation, 2023. 4.4

[60] OpenAI. Gpt-4 technical report, 2023. 4.4

[61] Ravi Pandya, Zhuoyuan Wang, Yorie Nakahira, and Changliu Liu. Towards proactive safe human-robot collaborations via data-efficient conditional behavior prediction. *arXiv preprint arXiv:2311.11893*, 2023. 2.1

[62] Joon Sung Park, Joseph C O'Brien, Carrie J Cai, Meredith Ringel Morris, Percy Liang, and Michael S Bernstein. Generative agents: Interactive simulacra of human behavior. *arXiv preprint arXiv:2304.03442*, 2023. 2.2, 2.3

[63] Shishir G Patil, Tianjun Zhang, Xin Wang, and Joseph E Gonzalez. Gorilla: Large language model connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023. 2.3

[64] Top Piriyakulkij, Volodymyr Kuleshov, and Kevin Ellis. Active preference inference using language models and probabilistic reasoning, 2023. 2.1

[65] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba. Virtualhome: Simulating household activities via programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8494–8502, 2018. 5.1

[66] Aryn A Pyke, Jon M Fincham, and John R Anderson. When math operations have visuospatial meanings versus purely symbolic definitions: Which solving stages and brain regions are affected? *NeuroImage*, 153:319–335, 2017. 2.2

[67] 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. 6.2

[68] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR, 2023. 6.2

[69] Fred Reed. Promise of ai not so bright. *The Washington Times*, 2006. 6.2

[70] Oscar J Romero, John Zimmerman, Aaron Steinfeld, and Anthony Tomasic. Synergistic integration of large language models and cognitive architectures for robust ai: An exploratory analysis. *arXiv preprint arXiv:2308.09830*, 2023. 2.2, 4

[71] Gabriel Sarch, Zhaoyuan Fang, Adam W Harley, Paul Schydlo, Michael J Tarr, Saurabh Gupta, and Katerina Fragkiadaki. Tidee: Tidying up novel rooms using visuo-semantic commonsense priors. In *European Conference on Computer Vision*, pages 480–496. Springer, 2022. 4.4

[72] Gabriel Sarch, Yue Wu, Michael J Tarr, and Katerina Fragkiadaki. Open-ended instructable embodied agents with memory-augmented large language models. *arXiv preprint arXiv:2310.15127*, 2023. 5.1

[73] Mohit Shridhar, Jesse Thomason, Daniel Gordon, Yonatan Bisk, Winson Han, Roozbeh Mottaghi, Luke Zettlemoyer, and Dieter Fox. Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10740–10749, 2020. 5.1

[74] Gautam Singh, Yi-Fu Wu, and Sungjin Ahn. Simple unsupervised object-centric learning for complex and naturalistic videos. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=eYfIM88MTUE. 3.1, 6.2

[75] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. ProgPrompt: Generating situated robot task plans using large language models. In *International Conference on Robotics and Automation (ICRA)*, 2023. URL https://arxiv.org/abs/2209.11302. 4, 4.4

[76] DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. *Advances in Neural Information Processing Systems*, 34:14502–14515, 2021. 2.1

[77] Theodore Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas Griffiths. Cognitive architectures for language agents. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=1i6ZCvflQJ. Survey Certification. 2.2

[78] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *arXiv preprint arXiv:2303.08128*, 2023. 2.3

[79] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 3.2

[80] Mirac Suzgun and Adam Tauman Kalai. Meta-prompting: Enhancing language models with task-agnostic scaffolding. *arXiv preprint arXiv:2401.12954*, 2024. 2.3

[81] Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021. 1

[82] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109. 5.1

[83] Brandon Trabucco, Gunnar A Sigurdsson, Robinson Piramuthu, Gaurav S. Sukhatme, and Ruslan Salakhutdinov. A simple approach for visual room rearrangement: 3d mapping and semantic search. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=1C6nCCaRe6p. 4.4

[84] Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. Chatgpt for robotics: Design principles and model abilities. Technical Report MSR-TR-2023-8, Microsoft, February 2023. URL https://www.microsoft.com/en-us/research/publication/chatgpt-for-robotics-design-principles-and-model-abilities/. 2.3

[85] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023. 2.3, 4.1, 4.2, 4.3, 4.4, **??**

[86] Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. Emergent abilities of large language models. *Transactions on Machine Learning Research*, 2022. 2.3

[87] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022. 2.3, 4.2

[88] Yaqi Xie, Chen Yu, Tongyao Zhu, Jinbin Bai, Ze Gong, and Harold Soh. Translating natural language to planning goals with large-language models. *arXiv preprint arXiv:2302.05128*, 2023. 4

[89] Xin Xin, Alexandros Karatzoglou, Ioannis Arapakis, and Joemon M Jose. Self-supervised reinforcement learning for recommender systems. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 931–940, 2020. 2.1

[90] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36, 2024. 2.3

[91] Sriram Yenamandra, Arun Ramachandran, Karmesh Yadav, Austin Wang, Mukul Khanna, Theophile Gervet, Tsung-Yen Yang, Vidhi Jain, Alexander William Clegg, John Turner, et al. Homerobot: Open-vocabulary mobile manipulation. *arXiv preprint arXiv:2306.11565*, 2023. 1

[92] Ce Zhang, Simon Stepputtis, Joseph Campbell, Katia Sycara, and Yaqi Xie. Hiker-sgg: Hierarchical knowledge enhanced robust scene graph generation. *arXiv preprint arXiv:2403.12033*, 2024. 3.1

[93] Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, Feng Yin, Yitao Liang, and Yaodong Yang. Proagent: Building proactive cooperative agents with large language models, 2024. 2.3, **??**

[94] Hongxin Zhang, Weihua Du, Jiaming Shan, Qinhong Zhou, Yilun Du, Joshua B Tenenbaum, Tianmin Shu, and Chuang Gan. Building cooperative embodied agents modularly with large language models. *arXiv preprint arXiv:2307.02485*, 2023. 2.3

[95] Jesse Zhang, Jiahui Zhang, Karl Pertsch, Ziyi Liu, Xiang Ren, Minsuk Chang, Shao-Hua Sun, and Joseph J Lim. Bootstrap your own skills: Learning to solve new tasks with large language model guidance. *arXiv preprint arXiv:2310.10021*, 2023. 2.3, **??**

[96] Michelle Zhao, Reid Simmons, and Henny Admoni. Coordination with humans via strategy matching. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9116–9123. IEEE, 2022. 2.1

[97] Z.H. Zhou and S. Liu. *Machine Learning*. Springer Nature Singapore, 2021. ISBN 9789811519673. URL https://books.google.com/books?id=ctM-EAAAQBAJ. 2.1

[98] Feiyu Zhu and Reid Simmons. Bootstrapping cognitive agents with a large language model, 2024. 1

[99] Andy Zou, Zifan Wang, J. Zico Kolter, and Matt Fredrikson. Universal and transferable adversarial attacks on aligned language models, 2023. 2.3

Appendix

A Comparison to Other Agent Frameworks

Table A.1 shows the comparison between the agent framework we proposed to some other contemporary agent frameworks.

	ACT-R [6]	BOSS [95]	ProAgent [93]	RT2 [18]	SayCan [3]	SOAR [38]	Voyager [85]	Lixr (ours)
Object-centric representation	✓	✓	✓	✓	✓	✓	✓	✓
Explicit procedural policy	✓	✓	✗	✗	✗	✓	✓	✓
Precondition function	Symbolic	Any	-	-	-	Symbolic	Any	Any
Episodic history	Implicit	None	Explicit	None	None	Explicit	Explicit	Explicit
Explicit declarative knowledge	✓	✗	✗	✗	✗	✓	✓	✓
Explicit affordance model	✓	✓	✓	✗	✓	✓	✓	✓
Unlimited working memory content	✗	✓	✗	✗	✗	✗	✓	✓
Multiple strategies for the same task	✓	✗	✗	✗	✓	✓	✗	✓
Image as direct input	✗	✗	✗	✓	✓	✗	✗	✗
Low-level action control	✗	✗	✗	✓	✗	✗	✗	✗

Table A.1: Comparison between Lixr and other agent frameworks

B Sample Environment Memory

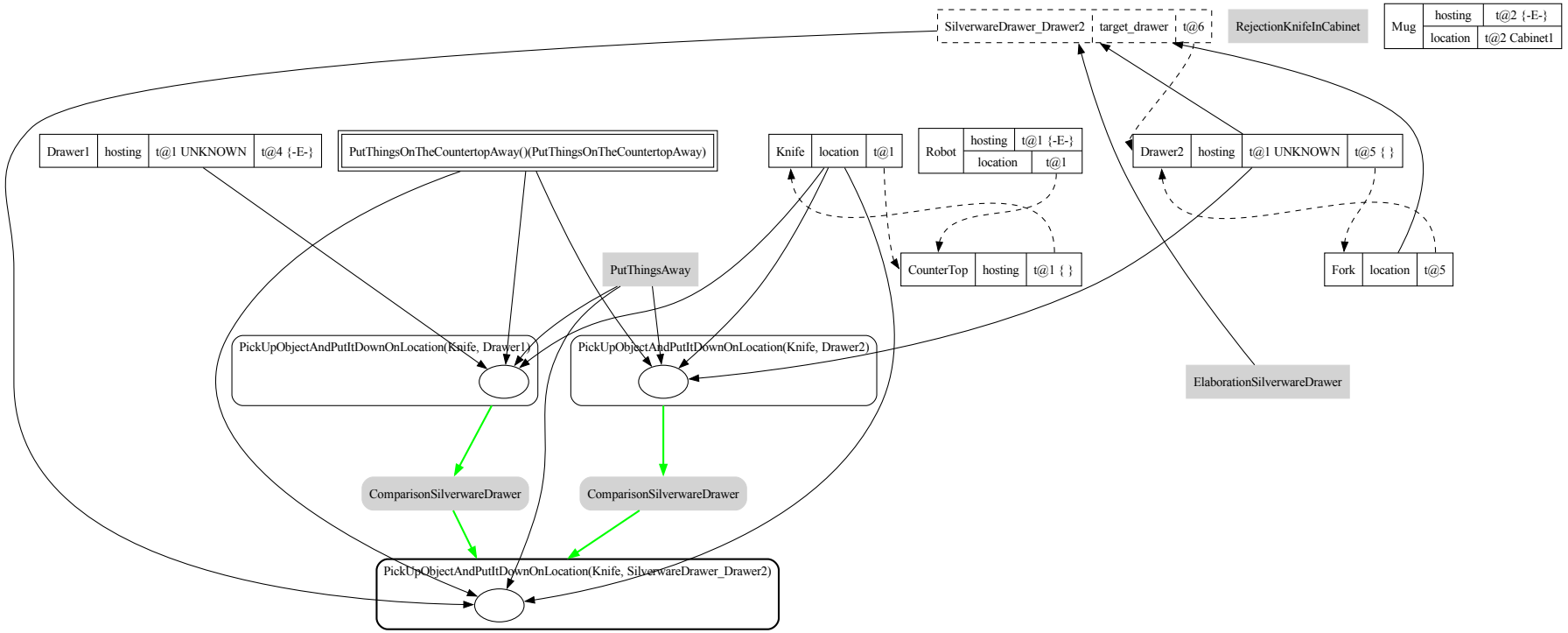


Figure B.1: Sample memory layout of an agent.

C Bootstrapping Examples

Action Prompting Example

Below is an example of the final action generated and its purpose:

```
[Option Suggestion]
"motor action: open Cabinet_5c8637ba"

[Purpose]
The purpose of the suggested option is to open the cabinet that the robot is
currently in front of, so it can be explored for an egg.
```

Production Prompting Example

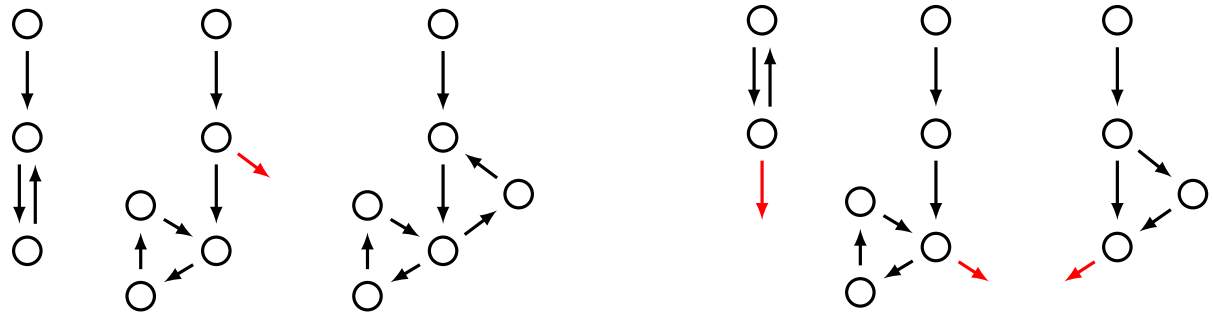
Below is an example of the English description generated for a production rule:

```
[Specific Rule]
IF the current task is to find an object AND the robot is in front of a closed
receptacle AND the robot's gripper is empty THEN choose motor action: open the receptacle.

[Generalizable Constants]
 * "Cabinet_5c8637ba" could be any closed receptacle that the robot is in front of.
 * "Egg" could be any object that the robot is tasked to find.

[Generalized Rule]
IF the current task is to find a/an <object> AND the robot is in front of a closed
<receptacle> AND the robot's gripper is empty THEN choose motor action: open <receptacle>.
```

Code Generated Example

Below is an example of the coded generated by for the same production rule as above (partially reformatted to fit the page):

```
class OpenReceptacle(Production):
    english_description = "IF the current task is to find a/an <object> "
      "AND the robot is in front of a closed <receptacle> "
      "AND the robot's gripper is empty "
      "THEN choose motor action: open <receptacle>."

    def precondition(
        self,
        current_task: str,
        current_location: str,
        previous_tasks: Dict[str, bool],
        spatial_knowledge: SpatialKnowledge,
        object_knowledge: ObjectKnowledge,
```



(a) cycle detection is triggered (b) cycle detection is not triggered

Figure C.2: Examples of transitions graphs for cycle detection. Each node represent a hash value of the environment knowledge of the agent (current or past), and the directed edges represent applicable production rules. A edge that points to nothing (marked by red) indicates that the corresponding production has never been executed and thus the agent does not have the knowledge of the state after applying the production.

```
) -> Tuple[bool, str]:

    # Parse the target object from the current task
    match = re.match(r"find a/an (\w+)", current_task)
    if not match:
        return (False, "The current task is not to find an object.")
    self.target_object = match.group(1)

    # Check if the robot is in front of a closed receptacle
    if not object_knowledge[spatial_knowledge[current_location]].opened == False:
        return (False, "The robot is not in front of a closed receptacle.")
    self.target_receptacle = current_location

    # Check if the robot's gripper is empty
    if not spatial_knowledge["RobotGripper"].hosting == []:
        return (False, "The robot's gripper is not empty.")

    return (True, "")

def apply(self) -> str:
    return f"motor action: open {self.target_receptacle}"
```

Cycle Detection Examples

Figure C.2 illustrates when cycle detection are triggered.

End Condition Examples

The following are some examples of the end condition generated for different tasks.

- `explore a/an <receptacle>`: "the robot has fully explored the receptacle."
- `find a/an <object>`: "the robot has found the object and has it in its gripper."
- `pick up and place a/an <object> in/on a/an <receptacle>`: "the robot has successfully picked up the specified object and placed it in/on the specified receptacle, and the robot's gripper is empty."
- `slice a/an <sliceable>`: "the sliceable object is already sliced and the robot's gripper is holding a knife."
- `put things on the countertops away`: "all objects on the countertops have been put away in the cabinets and there are no more unexplored countertops or cabinets."

Production Improvement Example

Below is an example of the LLM output for production improvement on the `find` task.

```
[Common Cases]
 * The object is in plain sight and the robot just needs to move to it and pick it up.
 * The object is inside a closed receptacle and the robot needs to open the receptacle
 before picking up the object.
 * The object is not in the robot's field of view and the robot needs to explore the
 kitchen to find it.

[Existing Productions]
 * "IF the current task is to find a/an <object> AND the robot's gripper has <object>
 THEN choose special action: 'done'": This production is reasonable and covers the end
 condition of the task. No changes needed.
 * "IF the current task is to find a/an <object> AND the <object> is located in
 <location> AND the robot is not at the location of the <object> AND the robot's
 gripper is empty THEN choose motor action: move to <location>": This production is
 reasonable but overly constrained because it only applies when the robot's gripper is
 empty. The robot can still move to the location of the object even if its gripper is
 not empty.
 Suggestion: change it to "IF the current task is to find a/an <object> AND the <object>
 is located in <location> AND the robot is not at the location of the <object> THEN
 choose motor action: move to <location>".
 [OTHER PRODUCTIONS OMITTED IN THE INTEREST OF SPACE]

[Uncovered Cases]
 * Case1: The robot's gripper is occupied and it needs to pick up the object
   - The robot is at the location of the object
   - The robot's gripper is not empty
 * Case2: The robot's gripper is occupied and it needs to open a receptacle
   - The robot is in front of a closed receptacle
   - The robot's gripper is not empty

[New Production Rule]
 * IF the current task is to find a/an <object> AND the <object> is located in a/an
```

```
<receptacle> AND the robot is in front of the <receptacle> AND the robot's gripper is
not empty THEN choose motor action: put down <object_in_gripper> and pick up <object>.
* IF the current task is to find a/an <object> AND the robot is in front of a closed
<receptacle> AND the robot's gripper is not empty THEN choose motor action: put down
<object_in_gripper> and open <receptacle>.
```

Prompt for Declarative Knowledge

Whenever the agent needs to verify the correctness of a statement in natural language, it makes the following query to the LLM and parse the output into a boolean value.

```
Evaluate the correctness of the following statement: {STATEMENT}, reply only True,
  False, or Unknown
```

D User Preferences Collection Details

Correct Update Examples

The following is GPT4's response on the preference "Have a glass of water and it needs to be cold."

```
[Feedback Classification]
Change in the breakfast menu

[Preference Expression]
 * add Task Proposing rule: IF the task is to 'make a Breakfast' THEN propose task 'serve
 a Glass of Cold Water'
 * add Task Proposing rule: IF the task is to 'serve a Glass of Cold Water' AND there is
 NOT a Glass on Table AND there is a Glass on Countertop THEN propose task pick and place
 Glass to Table
 * add Task Proposing rule: IF the task is to 'serve a Glass of Cold Water' AND there is a
 Glass on Table AND there is NOT Water in Glass THEN propose task apply Water Bottle to Glass
 * add Task Proposing rule: IF the task is to 'serve a Glass of Cold Water' AND there is a
 Glass on Table AND there is Water in Glass AND Water is NOT Cold THEN propose task apply
 Ice Cube to Glass
 * add Task Proposing rule: IF the task is to 'serve a Glass of Cold Water' AND there is a
 Glass on Table AND there is Water in Glass AND Water is Cold THEN propose task DONE: serve
 a Glass of Cold Water

[Existing Rule Modification]
 * "IF the task is to 'make a Breakfast' AND there is a Sandwich on Table AND a Cereal on
 Table THEN propose task DONE: make a Breakfast": Relevant to the user feedback and
 inconsistent - should be changed to "IF the task is to 'make a Breakfast' AND there is a
 Sandwich on Table AND a Cereal on Table AND there is a Glass of Cold Water on Table THEN
 propose task DONE: make a Breakfast"
 * All other rules: Irrelevant to the user feedback - leave unchanged

[End]
```