Collaborating with Ad Hoc Agents through Reasoning and Learning

Hasra Dodampegama[1], Mohan Sridharan[1]

[1]*Institute of Perception, Action and Behavior, School of Informatics, University of Edinburgh, UK*

Abstract

An assistive AI agent often has to collaborate with previously unseen agents and humans. Methods considered state of the art for such *ad hoc teamwork* use a large labeled dataset of prior observations to model the behavior of other agents and to determine the ad hoc agent's behavior. These approaches are resource-hungry, and do not support rapid incremental revisions or transparency, with the necessary resources (e.g., training examples, computation) not readily available in practical domains. Our architecture for ad hoc teamwork embeds the principles of refinement, ecological rationality, and interactive learning, leveraging the complementary strengths of knowledge-based and data-driven methods. For any given goal, an ad hoc agent determines its actions through non-monotonic logical reasoning with: (a) prior domain-specific commonsense knowledge; (b) models learned rapidly to predict the behavior of other agents; and (c) anticipated abstract future tasks based on generic knowledge of similar situations. Further, the agent processes natural language descriptions and observations of other agents' behavior, incrementally acquiring and revising its existing knowledge. We evaluate the capabilities of our architecture in *VirtualHome*, a realistic 3D simulation environment.

Keywords

Knowledge representation, Non-monotonic logical reasoning, Ecological rationality, Ad hoc teamwork

1. Introduction

Consider an AI agent performing household tasks in collaboration with other agents (AI, human) it has not worked with before. Figure 1 shows snapshots of this motivating scenario in which an AI agent (blue shirt) and a human (green top) are preparing breakfast and setting up a workstation. The AI agent has to reason with different descriptions of prior knowledge and uncertainty in the form of qualitative statements ("eggs are usually in the fridge") and quantitative statements ("I am 90% sure I saw the eggs on the table"). The agents have a limited view of the environment and do not communicate with each other, although each of them is aware of the domain state and action outcomes, e.g., the location of teammates and objects moved by a teammate. Furthermore, the human may want to query and understand the AI agent's decisions. These characteristics correspond to *Ad hoc Teamwork* (AHT), requiring cooperation "on the fly" without prior coordination [1]. The AHT problem arises in many practical applications such as disaster rescue, exploration, and collaborative games, and poses challenges in knowledge representation, reasoning, and learning [2].

Figure 1: Screenshots from *VirtualHome* [3] showing a human (female in green top) and an assistive AI agent (male in blue shirt) collaborating.

C.A.R.L.A.: Workshop on Cognitive Architectures for Robotics: LLMs and Logic in Action FLoC, July 18, 2026, Lisbon, Portugal.
hasra.dodampegama@ed.ac.uk (H. Dodampegama); m.sridharan@ed.ac.uk (M. Sridharan)
0000-0003-2302-1501 (H. Dodampegama); 0000-0001-9922-8969 (M. Sridharan)

The state of the art in AHT has moved from using predetermined policies for selecting actions in specific states to methods with a key *data-driven* component that uses a long history of prior experiences to build probabilistic or deep network methods that model the behavior of other agents (or agent types) and optimize the behavior of the ad hoc agent [2]. However, it is difficult to gather training samples of different situations in practical domains. Also, these methods are usually opaque, offering limited insight into how or why an agent made a decision. In a departure from existing work, we describe an architecture, KAT, that leverages the complementary strengths of knowledge-based and data-driven methods for reasoning and learning, enabling each ad hoc agent in a team to:

1. Adapt scalably through non-monotonic logical reasoning with prior commonsense domain knowledge and rapidly-learned predictive models of other agents' behavior;
2. Leverage a Large Language Model (LLM) to anticipate future tasks to be completed, adapting the LLM's output based on domain-specific knowledge and experience; and
3. Incrementally revise prior knowledge based on LLM-based processing of natural language descriptions of actions and outcomes, and decision tree induction applied to observations.

We have published subsets of these capabilities in other papers, starting with a proof of concept architecture for reasoning [4], and extending it to support explainability [5], knowledge revision and scalability [6]. Here, we bring these component together further highlighting the benefits of the interplay between reasoning and learning. We evaluate the architecture's capabilities in *VirtualHome*, a realistic 3D simulation environment [3], with simulated agents collaborating to perform household tasks.

2. Related Work

Researchers have explored AHT for more than two decades under different names [2]. Initial work used predefined protocols or plays that encoded specific actions for the ad hoc agent in specific scenarios [7]. Subsequent work used probabilistic and sampling-based methods such as Upper Confidence bounds for Trees to determine the ad hoc agent's actions [8, 9]. Recent approaches considered state of the art for AHT have posed it primarily as a learning problem; the key component predicts behavior of other agents and determines the ad hoc agent's behavior using a history of prior interactions with similar agents or agent types. This includes the use of Fitted Q Iteration to learn action selection policies from offline data of each teammate type [10]; attention-based recurrent neural networks for real-time adaptation [11]; graph neural networks to simulate interactions with other agents (agent types) and determine the ad hoc agent's behavior [12]; self-play to learn a cooperation policy and candidate teammate policies [13]; sequential and hierarchical variational auto encoders to model teammates' changing behaviors [14]; and model-based RL methods to learn separate models of the environment and teammates [15]. Also, an LLM-based framework has been developed to compute the ad hoc agent's actions [16].

While the frameworks summarized above offer promising strategies for modeling the behavior of teammates and selecting the actions of the ad hoc agent, they are resource-hungry, requiring substantial time, computation, and training examples to achieve the observed performance. Such resources are not readily available in practical AHT domains, posing questions about the scalability and usability of these methods. Also, they struggle to leverage the commonsense knowledge available in many practical domains, and their internal decision-making mechanisms are often opaque, limiting the ability to understand, justify, and audit the agents' automated decisions. This paper builds on our prior work [4] that enabled an ad hoc agent to reason with domain knowledge and predictive models of other agents in simple domains (e.g., soccer). Here we consider a more complex household domain, enabling the agent to leverage an LLM to anticipate high-level future tasks, and use decision-tree induction and an LLM for acquiring previously unknown knowledge from observations and natural language descriptions.

3. Architecture

Example Domain 1. *[Example Embodied AI Agent Domain]*
Consider an embodied AI agent and a human agent collaborating to complete household tasks; Figure 1 shows snapshots while preparing breakfast [3]. The agents can interact with the environment through

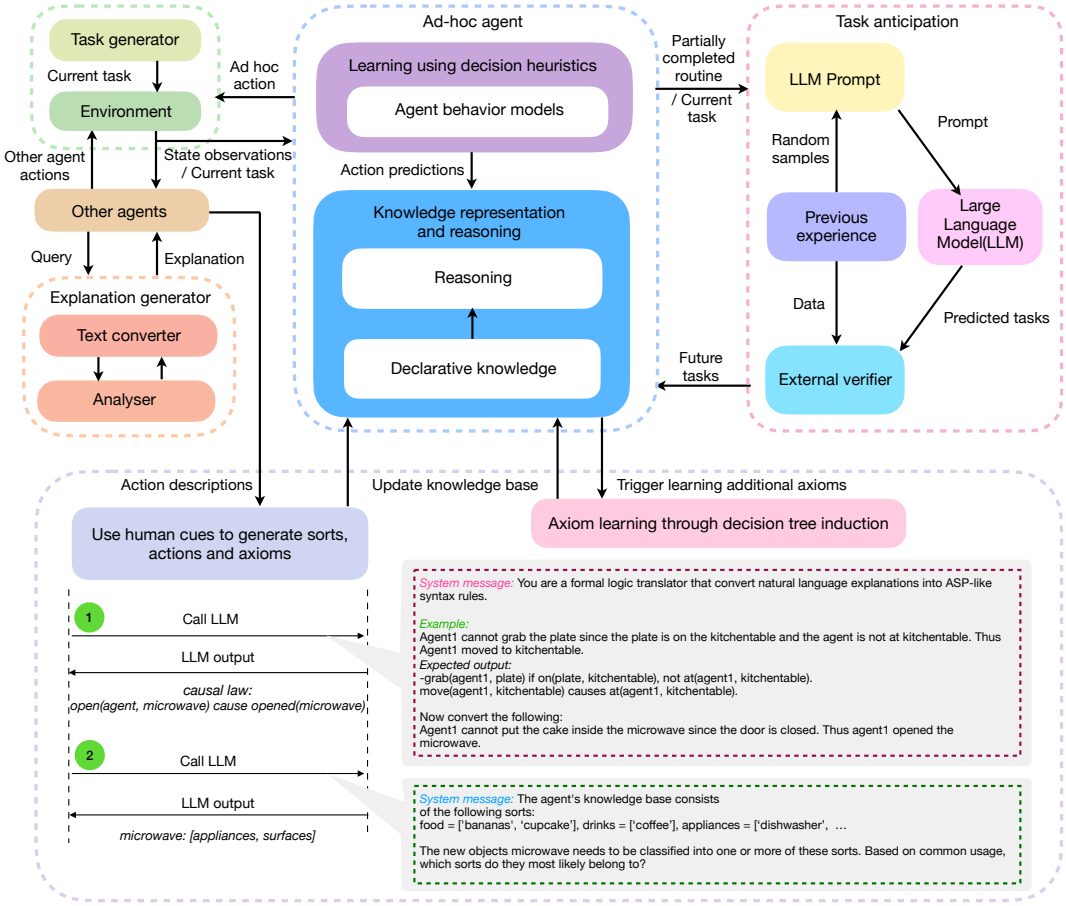

Figure 2: KAT leverage the complementary strengths of knowledge-based and data-driven reasoning and learning.

high-level actions, e.g., move to places, pick up or place objects, switch appliances on or off, and open or close appliances. Completing a task requires a sequence of such actions to be computed and executed by the agents who do not communicate directly with each other. Also, the agent assumes that the other agents have access to the same state information and will make (what it considers as) rational decisions and computes its plan to complete the current task and prepare for the upcoming task(s). Prior commonsense knowledge of the embodied agent includes relational descriptions of some attributes of the domain, ad hoc agent, and the human agent (Section 3.1); default statements that hold in all but a few exceptional circumstances; and some axioms governing actions and change, e.g., the agent or human can only pick one object at a time, and certain food items require preparation prior to consumption.

3.1. Knowledge Representation and Reasoning

In KAT, the transition diagram of any given domain is described using an extension of action language $\mathcal{AL}_d$ [17]. Action languages are formal models of parts of natural language for describing transition diagrams of dynamic systems. The domain representation comprises a system description $\mathcal{D}$, a collection of statements of $\mathcal{AL}_d$, and a history $\mathcal{H}$. $\mathcal{D}$ has a sorted signature Σ with basic sorts, and domain attributes (statics, fluents) and actions described in terms of these sorts. Basic sorts in our example scenario include $object$, $appliance$, ad_hoc_agent, $human$, and $step$ (for temporal reasoning), and are arranged hierarchically, e.g., $apple$ is a sub-sort of $food$, a sub-sort of $graspable$, a sub-sort of $object$. Actions can be $agent\_actions$ or $exo\_actions$ (exogenous actions). The $agent\_actions$ such as $grab(ad\_hoc\_agent, object)$ are performed by the ad hoc agent. The $exo\_actions$ such as $exo\_grab(other\_agent, object)$ are performed by other agents, e.g., human or another AI agent. Statics are domain attributes whose values cannot change. Fluents are attributes whose values can change; they can be *inertial*, which obey inertia laws and are changed by the ad hoc agent's actions, e.g.,

$at(ad\_hoc\_agent, location)$ is the ad hoc agent's location, or *defined*, which do not obey inertia laws and are not directly changed by the ad hoc agent's actions, e.g., $agent\_at(other\_agent, location)$ is a teammate's location computed by (say) external sensors.

Given Σ, the domain's dynamics are described in $\mathcal{D}$ using three types of axioms: *causal law*, *state constraint*, and *executablility condition* such as:

$$grab(A, O) \textbf{ causes } in\_hand(A, O) \tag{1a}$$

$$heated(F) \textbf{ if } on(F, E), switched\_on(E) \tag{1b}$$

$$\textbf{impossible } grab(A, O) \textbf{ if } on(O, E), not\ opened(E) \tag{1c}$$

where Statement 1(a), a causal law, implies that grabbing an object causes it to be in the hand of the ad hoc agent; Statement 1(b), a state constraint, implies that a food item placed in an electrical appliance (e.g., microwave) that is switched on gets heated; and Statement 1(c), an executability condition, prevents the ad hoc agent from trying to grab an object from an appliance with a closed door. History $\mathcal{H}$ is a record of observations of the form $obs(fluent, boolean, step)$, and action executions of the form $hpd(action, step)$ at specific time steps. It also includes default statements that are true in all but a few exceptional circumstances, e.g., books are usually in the library but cookbooks are in the kitchen.

To reason with knowledge, KAT uses a script to automatically construct program $\Pi(\mathcal{D}, \mathcal{H})$ by translating the system description $\mathcal{D}$ and history $\mathcal{H}$ to CR-Prolog [18], an extension to ASP that supports consistency restoring (CR) rules. $\Pi(\mathcal{D}, \mathcal{H})$ includes statements from $\mathcal{D}$ and $\mathcal{H}$, inertia axioms, reality check axioms, closed world assumptions for defined fluents and actions, helper relations, e.g., $holds(fluent, step)$ and $occurs(action, step)$ to imply that a fluent is true and an action is part of a plan at a time step, and helper axioms that define goals and guide planning and diagnosis. ASP encodes *default negation* and *epistemic disjunction*, and supports non-monotonic logic reasoning. This ability to revise previously held conclusions is essential for agents reasoning and acting in practical domains based on incomplete knowledge and noisy observations. The CR rules allow the agent to make assumptions (e.g., that a default statement does not hold) under exceptional circumstances to recover from inconsistencies. All reasoning tasks (i.e., planning, diagnostics, and inference) are then reduced to computing *answer sets* of Π. We use the SPARC system [19] to solve CR-Prolog programs. Example programs and results are in our open source repository [20].

Our current application domain is substantially complex with many objects and actions making the corresponding domain description more complex and the tasks requiring much longer plans. To support scalability, we build on prior work on a refinement-based architecture [21]. Specifically, we enable the ad hoc agent to formally define a fine-resolution description of the domain (e.g., with regions and object parts) as a refinement of a coarse-resolution description (e.g., with rooms and objects), with the agent now able to reason about different aspects of the domain as needed.

3.2. Agent Behavior Models

Reasoning with just prior domain knowledge that can be incomplete or inconsistent will lead to poor performance, particularly under AHT settings where the other agent's actions also revise the domain state. KAT enables the agent to reason with models that predict the action choices of other agents and are learned (and revised) rapidly. This is achieved by embedding the principle of *Ecological Rationality* (ER) [22], which is based on Herb Simon's original definition of Bounded Rationality [23] and the algorithmic theory of decision heuristics [24]. ER explores decision making "in the wild", i.e., under open world uncertainty with the space of possibilities not fully known a priori, and characterizes behavior as a joint function of internal cognitive processes and the environment. It advocates the use of *adaptive satisficing* for making decisions in open worlds: in the absence of comprehensive knowledge, optimal decisions may be unknowable and not just hard to compute. Thus, decision heuristics (e.g., tallying, sequencing, fast and frugal methods) are used to ignore part of the information, and to make decisions more quickly, frugally, and accurately [24] often achieving better performance than sophisticated methods with many more free parameters in practical applications [25].

Table 1: Attributes used to create the behavior models of the other agents in VirtualHome.

Description of the attribute
Immediate two previous actions of the agent
Position of the agent (x,y,z)
Orientation of the agent (x,y,z)
Objects associated with the goal
Any objects in the hand of the agent
Any objects in the hand of the remaining agents
Current and previous tasks
Flags (weekday, going to office, guests expected)

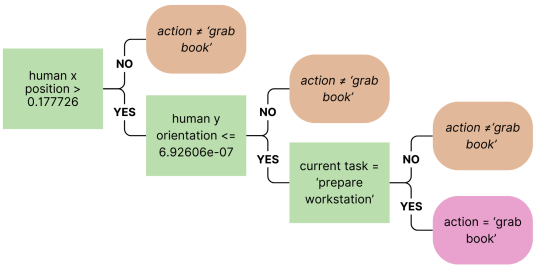

Figure 3: FF tree model of human behavior for the grab_book action in the VirtualHome domain.

Specifically, KAT enables the ad hoc agent to select relevant attributes and learn models of the other agents behavior incrementally and from limited data. The agent learns an ensemble of *Fast and Frugal* (FF) trees to predict the behavior of each teammate (or teammate type). Each FF tree provides a binary choice for a particular action, and the number of leaves in a tree is limited by the number of attributes [26]. Each level of the tree contains an exit allowing the agent to make quick decisions based on available data by evaluating the more informative attributes and stopping as soon as a rational option is found. Figure 3 shows an FF tree learned for a human, with Table 1 showing the key attributes used in these trees. Since each FF tree provides a binary choice, we build each predictive model as an ensemble of FF trees with a simple decision tree choosing an action based on the output of the FF trees. As we show later (Section 4.2), reasoning with prior knowledge and these models provides much better performance than methods that just reason with knowledge or use learned models. Consistent agreement (or disagreement) between actual observations and the behavior models predictions triggers the use of a particular model for subsequent steps, or revision of the existing model(s), allowing the ad hoc agent to quickly adapt to changes in the domain or another agent's behavior over time.

3.3. Task Anticipation

An agent's ability to anticipate future tasks and goals in the domain can enhance collaboration and overall performance. However, encoding such knowledge manually across different domains can be challenging and impractical. On the other hand, LLMs have been shown to be effective when used to translate natural language to domain-specific language, and to generate high-level guidance that is implemented by planning subroutines [27]. We thus enable an ad hoc agent to use an LLM to anticipate the high-level future task (e.g., *prepare dinner*) likely to be assigned once the current task is done. We experimentally demonstrate that jointly planning to complete the current and anticipated tasks improves team performance. The agent uses a combination of prompting strategies with the LLM:

- **Adopting persona:** A specific role or character is assigned to guide the LLM's responses to be more (contextually) consistent with the assigned role.
- **Few-shot prompting:** The prompt includes a few examples of the expected output in specific situations, guiding the use of pretrained knowledge.
- **Chain-of-thought (CoT):** The prompt includes a step-by-step reasoning process that can be followed to arrive at an answer, leading to more accurate responses.

A 'system message' guides the LLM to adopt the persona of a household assistant and complete the partial task routine (provided as input) by selecting from the available list of tasks in the example scenario (in *VirtualHome*). With 'few-shot' prompting, the prompt includes two task routines randomly chosen from previous days. Next, CoT prompting is used to explain the reasoning behind each task in these example task routines. Such explanations can be provided by the system designer or generated by the LLM. The system message, few-shot examples with CoT explanation, and the current query (i.e., partially completed or empty task routine for the day) are provided as input to the LLM.

The LLM's output to a prompt is parsed by an *external validator* to check whether the tasks are feasible and in a reasonable order. Specifically, the validator compares the LLM's output with domain-

and task-specific features extracted from existing knowledge and recent observations, eliminating tasks that are invalid or irrelevant and reordering tasks according to human preferences. For example, the agent will prioritize preparation of the workstation over packing a lunch box when the human is working from home. Since the list of validated tasks can change over time, the ad hoc agent considers one anticipated task and the current task as the joint goal for which a plan is computed.

3.4. Knowledge Acquisition

Since decisions based on incomplete or inconsistent knowledge can lead to ineffective collaboration, KAT enables the ad hoc agent to incrementally acquire domain knowledge and reduce ambiguity. The agent acquires knowledge using two strategies: (i) it learns from cues provided by a human during task execution; and (ii) it explores actions in different states, identifying and correcting inconsistencies.

Learning from human cues: When the ad hoc agent receives a cue from the human (e.g.,"Agent 1 cannot put the cake inside the microwave since the microwave's door is closed"), it automatically generates and sends a prompt to the LLM by combining a predefined system message (message 1 in Figure 2) with two examples of input and expected output for few-shot prompting, the structure of the two types of axioms (e.g., "a causes f if p"), and the query based on the cue (Line 1, Algorithm 1). The LLM then maps these cues into structured logic rules (candidate axioms) using their known structure. The output is then parsed using regular expressions to discard any that does not match the expected format of axioms. From the remaining output, the agent extracts actions (verbs), objects, and action preconditions and effects by mapping them to causal laws or executability conditions using known axiom templates (Line 2). The agent then examines its current Answer Set Prolog program, ($\Pi(\mathcal{D}, \mathcal{H})$), to verify whether the objects from the cue are already defined (Line 3). Any new objects are returned to the LLM with a second prompt (message 2 in Figure 2) based on a predefined message template, existing sorts in $\Pi(\mathcal{D}, \mathcal{H})$, and the new objects, asking the LLM to assign a sort label to the new objects (Lines 4-6). The LLM categorizes the new objects into one or more of the known basic sorts. If multiple sort labels are returned for an object, the lowest category in the sort hierarchy is used (Line 7) and the new object is added to the Answer Set Prolog program under that sort category (Line 8).

Next, the agent examines its current knowledge base ($\Pi(\mathcal{D}, \mathcal{H})$) to identify whether the action from the human cue is already defined (Line 10). Since the same action can be defined in multiple ways, the agent extracts the action verb (e.g., *grab*) and checks whether it exists in the knowledge base. If an action that semantically matches this action verb exists in $\Pi(\mathcal{D}, \mathcal{H})$, the agent retrieves it; this action may exhibit the same behavior as the one in the cue but have a different name, e.g., *pick(A,O)* instead of *grab(A,O)*. The agent performs strictly controlled verb synthesis with WordNet [28] to retrieve synonyms for the action verb from the cue and checks the synonyms with the Σ of $\Pi(\mathcal{D}, \mathcal{H})$. If a match is found, the action verb from the LLM output is replaced by the existing action in Σ. Also, the sorts of the arguments of the action extracted from the human cue may differ from those of the action in $\Pi(\mathcal{D}, \mathcal{H})$, e.g., they may be subsorts or parent sorts. If the sorts of arguments of the action in $\Pi(\mathcal{D}, \mathcal{H})$ are parent sorts of those in the cue, the agent uses the existing action as is; if not, the agent lifts the sort of the existing action's arguments to the new sorts and updates $\Pi(\mathcal{D}, \mathcal{H})$ (Lines 12-14). If the extracted action (and any equivalent) does not exist in $\Pi(\mathcal{D}, \mathcal{H})$, the agent adds it with the lowest (i.e., most specific) sort labels for arguments (Lines 15-17). This process ensures that we do not introduce the same action multiple times with different sorts as its arguments. This procedure is also repeated for literals (Lines 19-29), and the actions and literals are used to convert the extracted axioms to the appropriate format, e.g., 'a causes l' converted to '*holds(f,I+1) :- occurs(a,I)*' (Line 31). The learned axioms are then added to $\Pi(\mathcal{D}, \mathcal{H})$ (Line 32-34) and used for subsequent reasoning.

Learning from observations. The second strategy enables the ad hoc agent to refine its knowledge based on observations obtained while using the existing knowledge for planning and execution. It extends and adapts existing ideas on combining *decision tree induction* with knowledge-based reasoning [29]. At the end of each iteration, the agent selects an action a_I from the newly learned set of actions A, and an initial state of the environment S_I, and simulates the execution of a_I in S_I (Lines 2-3, Algorithm 2). It then collect information about the outcomes (e.g., end state, presence or absence of an

Algorithm 1: Learning From Cues

Input: cue: Human cue; $\Pi(\mathcal{D}, \mathcal{H})$: Answer Set Prolog program; LLM: Pretrained LLM; $systems\_msg_1$: Predefined system msg1 for prompt; $system\_msg_2$: Predefined system msg2 for prompt; $examples$: For few shot prompting.

Output: $\Pi(\mathcal{D}, \mathcal{H})$: Updated Answer Set Prolog program.

1 $axiom \leftarrow LLM(system\_msg_1, examples, cue)$
2 $action, literals, objects \leftarrow axiom$
3 $new\_objects \leftarrow \{o \in objects \mid o \notin \Pi(\mathcal{D}, \mathcal{H})\}$
4 **if** $new\_objects \neq \emptyset$ **then**
5 $\quad Sorts \leftarrow$ extractSorts($\Pi(\mathcal{D}, \mathcal{H})$)
6 $\quad sort\_categories \leftarrow LLM(system\_msg_2, Sorts, new\_objects)$
7 $\quad obj\_categories \leftarrow \{$ lowestSort$(o, sort_categories) \mid o \in new_objects \}$
8 $\quad \Pi(\mathcal{D}, \mathcal{H}) \leftarrow \Pi(\mathcal{D}, \mathcal{H}) \cup \{(o, obj\_categories[o]) \mid o \in new\_objects\}$
9 **end**
10 $action\_exists, existing\_action \leftarrow$ checkExists($action, \Pi(\mathcal{D}, \mathcal{H})$)
11 $Sorts \leftarrow$ extractSorts($\Pi(\mathcal{D}, \mathcal{H})$)
12 **if** $action\_exists$ with low level sorts **then**
13 $\quad action \leftarrow$ updateAction($action, existing\_action, Sorts$)
14 $\quad \Pi(\mathcal{D}, \mathcal{H}) \leftarrow \Pi(\mathcal{D}, \mathcal{H}) \cup action$
15 **else if** $\neg action\_exists$ **then**
16 $\quad action \leftarrow$ buildAction($action, Sorts$)
17 $\quad \Pi(\mathcal{D}, \mathcal{H}) \leftarrow \Pi(\mathcal{D}, \mathcal{H}) \cup action$
18 **end**
19 $lit\_exists, existing\_literals \leftarrow$ checkExists($literals, \Pi(\mathcal{D}, \mathcal{H})$)
20 **for** $literal \in literals$ **do**
21 $\quad$ **if** lit_exists with low level sorts **then**
22 $\quad\quad literal \leftarrow$ updateLiteral($literal, existing\_literals, Sorts$)
23 $\quad\quad \Pi(\mathcal{D}, \mathcal{H}) \leftarrow \Pi(\mathcal{D}, \mathcal{H}) \cup literal$
24 $\quad\quad existing\_literals \leftarrow existing\_literals \cup literal$
25 $\quad$ **else if** $\neg lit\_exists$ **then**
26 $\quad\quad literal \leftarrow$ buildAction($literal, Sorts$)
27 $\quad\quad \Pi(\mathcal{D}, \mathcal{H}) \leftarrow \Pi(\mathcal{D}, \mathcal{H}) \cup literal$
28 $\quad\quad existing\_literals \leftarrow existing\_literals \cup literal$
29 $\quad$ **end**
30 **end**
31 $axiom \leftarrow$ buildAxiom($axiom, action, existing\_literals$)
32 **if** *axiom* $\notin \Pi(\mathcal{D}, \mathcal{H})$ **then**
33 $\quad \Pi(\mathcal{D}, \mathcal{H}) \leftarrow \Pi(\mathcal{D}, \mathcal{H}) \cup axiom)$
34 **end**

inconsistency). In the presence of an inconsistency the agent triggers learning with the selected action (Lines 4-7) (a missing expected outcome indicates the potential absence of an executability condition(s); and observation of additional effects other than what is expected, indicates a missing causal law(s)). If all observations match expectations for different actions and states, the agent considers the current knowledge to be comprehensive. If an inconsistency is identified the agent simulates the execution of a_I in different states and stores the observed state transition information. It then constructs different version of the selected action a_i by using different sorts from the same group (Line 14), selects an initial state S_i (Line 15) and simulates their effects (Line 16). The agent then stores all fluent literals in the answer set at initial state, with an object constant that is also in a_i (Line 17). The agent repeats this procedure until sufficient data are gathered.

Algorithm 2: Learning From Observations

Input: $\mathcal{A}$: newly learned actions; $\Pi(\mathcal{D}, \mathcal{H})$: Answer Set Prolog program; $\mathcal{N}$: number of training examples; $\mathcal{K}$: number of iterations; $threshold$: confidence level.

Output: $\Pi(\mathcal{D}, \mathcal{H})$: Updated Answer Set Prolog program.

```
1  while i < 50 do
2      Select a_I ∈ A, S_I
3      S ← Simulate(a_I, S_I)
4      if S is not consistent with expectations then
5          trigger_learning ← True
6          a ← a_I
7          exit
8      end
9  end
10 if trigger_learning then
11     unobserved = ∅, unexpected = ∅
12     Sorts ← extractSorts(Π(D, H))
13     while unobserved < N or unexpected < N do
14         a_i ← constructAction(a, Sorts)
15         select S_i
16         S ← Simulate(a_i, S_i)
17         unobserved, unexpected ← extractFluents(S)
18     end
19     if unobserved > N then
20         Axioms ← DecisionTreeInduction(unobserved, K, threshold)
21     end
22     if unexpected > N then
23         Axioms ← DecisionTreeInduction(unexpected, K, threshold)
24     end
25 end
26 Π(D, H) ← Π(D, H) ∪ Axioms
27 return Π(D, H)
```

The agent constructs separate decision trees for learning causal laws and executability conditions (Lines 19-24). It replaces the ground terms in the collected *training* examples by their respective variables and reformats the dataset with the fluent literals as features and the presence or absence of inconsistency as the class label (Lines 1-2, Algorithm 3. Each training example then records the occurrence or non-occurrence of a fluent literal, and the presence or absence of an inconsistency, as a binary value. The agent then splits the processed data set into training and testing and uses the training data set to learn a decision tree by splitting the nodes using features that have not been used before and are likely to result in the highest reduction in entropy (Lines 6-7). It then uses the learned tree to classify the test data set and for each example records the sequence of decision tree nodes traversed from the root to the leaf node (Line 8). From the recorded paths it uses those whose leaf node's dominant class is inconsistent and agrees with their class label up to a threshold level and contains at least a minimum percentage of the dataset to generate candidate axioms by taking all possible combinations of the nodes appearing in each selected path (Lines 9-11). From the resulting candidates, it selects the candidates who have sufficient support among the test examples (Lines 13-14). It also repeats the decision tree induction process multiple times and adds only the axioms retained over multiple iterations to $\Pi(\mathcal{D}, \mathcal{H})$ (Line 26).

Figure 4 shows part of a decision tree generated by this process, leading to learning the following two executability conditions by the ad hoc agent which imply that an agent cannot switch on an appliance

Algorithm 3: Decision Tree Induction

Input: $\mathcal{F}$: Relevant literals; $\mathcal{K}$: Number of iterations; $threshold$: Confidence level.
Output: $Axioms$: Candidate axioms.

$\mathcal{F} \leftarrow$ replaceVariables($\mathcal{F}$)
$data \leftarrow$ preProcess($\mathcal{F}$)
$Axioms \leftarrow \emptyset$
for $k = 1$ *to* $\mathcal{K}$ **do**
  $cands \leftarrow \emptyset$
  $training, test \leftarrow data$
  $Tree \leftarrow$ buildDecisionTree($training$)
  $cand\_paths \leftarrow$ extractPaths($Tree, test$)
  for $c \in cand\_paths$ **do**
    if *leaf_support*(c) $\geq threshold$ ***and*** *min_samples*(c) ≥ 0.02 ***and*** *leaf_dom_class*(c) $=$ *inconsistent* **then**
      $\mathcal{C} \leftarrow$ generateCombinations(c)
      for $c' \in \mathcal{C}$ **do**
        if *test_sample_support*(c') $\geq threshold$ **then**
          $cands \leftarrow cands \cup c'$
        end
      end
    end
  end
  $Axioms \leftarrow Axioms \cup cands$
end
$Axioms \leftarrow \{ r \in Axioms \mid \text{freq}(r, Axioms) > n \}$
return $Axioms$

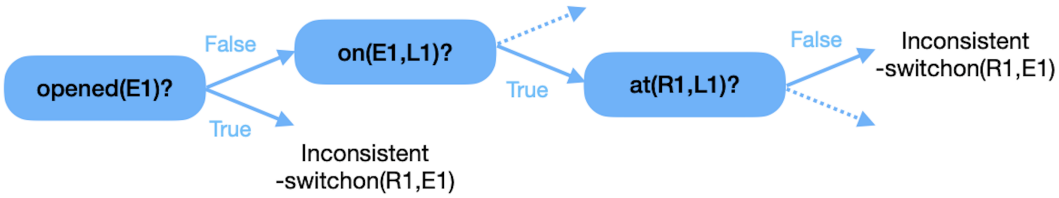

Figure 4: Part of the decision tree created to learn missing executability conditions.

if it is not in the same location or if the appliance's door is open.

$$-occurs(switchon(R, E), I) \leftarrow holds(on(E, L), I), not\ holds(at(R, L), I). \quad (2a)$$

$$-occurs(switchon(R, E), I) \leftarrow holds(opened(E), I). \quad (2b)$$

4. Experimental Setup and Results

We experimentally evaluated the following hypotheses regarding KAT's capabilities:

- **H1** Reasoning with prior knowledge and rapidly-learned behavior models improves performance;
- **H2** Using task anticipated by LLM as joint goal improves performance compared with not planning for joint goal;
- **H3** Incrementally-updated prompts and validators improve task anticipation and team performance;
- **H4** Using LLM to directly output low-level actions to complete tasks results in poor performance;
- **H5** KAT enables the ad hoc agent to accurately learn unknown objects, actions, and axioms; and
- **H6** Reasoning with incrementally learned knowledge improves the performance of the team.

We evaluated these hypotheses in the *VirtualHome* and recorded the number of steps (plan length) and the total time taken to complete the task(s) as the *performance measures*.

4.1. Experimental Setup

In the VirtualHome domain, we modeled the human as a simulated entity that chooses its actions based on an ASP program that considered prior knowledge and runtime observations but did not have models predicting teammates' actions. The human's ASP program also included actions such as eating and drinking that were not available to the ad hoc agent and encoded priorities and preferences, e.g., when preparing breakfast, the human toasted the bread before preparing the cereal. The sequence of tasks generated by the task generator were assigned to all agents (including the human agent) one at a time, and they were not aware of the complete task sequence. All agents received the same observations of the domain at each step, which they used to plan their respective actions without direct communication with each other. When an ad hoc agent equipped with KAT received a task, it prompted the LLM (Section 3.3). The anticipated tasks were validated and mapped to ASP literals, with the next anticipated task and current task set as joint goals for this agent. During planning, the ad hoc agent also used the learned behavior prediction model to predict each teammate's actions for a few steps (Section 3.2). These predictive models were built using only 1000 examples of prior traces of actions and domain state and were incrementally revised based on teammates observed behavior. The ASP program of the ad hoc agent included additional axioms for reasoning about these predicted actions of each teammate, which were mapped to exogenous actions. The ad hoc agent's plan anticipated preconditions of some intermediate steps to be satisfied by a teammate's actions, even though the teammate did not always execute that action. The ad hoc agent hence had to respond to unexpected action outcomes.

For evaluating **H1** and **H2**, in **Exp1**, we randomly selected 100 task routines sampled from predefined sequences and measured the ability of a team (human and an ad hoc agent) to complete these tasks. Performance measures were the number of steps and time taken. We used three baselines: **Base1**: LLM for anticipating future tasks, but no behavior models to predict human's actions; **Base2**: no LLM to anticipate future tasks, but behavior models to predict the human's actions; and **Base3**: did not use LLM for task anticipation or behavior models to predict human's actions. In the absence of a framework that supports all (or most) of KAT's capabilities, these baselines allowed us to conduct ablation studies and evaluate the contribution of each key component. Since the time taken and the number of actions in a plan can vary substantially based on the task, the average of these values across trials may not be meaningful. We instead *ran paired trials and computed performance measure values for baselines as a fraction of these values for KAT in each trial.* We then reported the average of these ratios.

To evaluate **H3**, in **Exp2**, we computed the precision and recall of tasks anticipated by the LLM before and after applying the validator, and a simple statistical model that replaced the LLM and anticipated the next task based on past experience, over 100 task routines. Further, in **Exp3**, we randomly selected 20 task routines and recorded the LLM's performance with and without prompting methods, using four baselines: **Base4**: no prompting strategy or validator; **Base5**: few-shot prompting but no validator; **Base6**: CoT prompts but no validator; and **Base7**: validator but no prompt-engineering.

For evaluating **H4**, with the LLM being used to directly compute sequences of actions for specific tasks (**Base8**), the prompt included details of actions available in our example scenario in *VirtualHome* and their intended purpose; some **Action Feasibility Rules**: (a) *movement limitation* (critical): must only move to adjacent locations defined by the next_to relationships; (b) *object location*: must be in the same location as an object to act on it (e.g., grab, put); (c) *carry limit*: cannot hold more than two objects, and actions like open, close, switch-on, or switch-off require putting at least one object down if holding two objects; (c) *appliance safety*: you should not open appliance doors when they are switched on; (d) *avoid conflict*: a human holding an object will perform actions involving object, so focus on other parts of the goal; current state: location of agents, objects, and appliances, and each appliance's state; description of the task to be performed; immediate prior actions of the human and ad hoc agent; other relevant information (e.g., human working from home); and a detailed example of selecting an action. It then asked the LLM to generate an action sequence for the ad hoc agent to complete the task. Feedback

Table 2
Average steps and completion time; values for baselines computed as a fraction of these values for KAT.

Architecture	*Steps*	*Time(s)*
KAT (anticipate tasks, predict actions)	1.0	1.0
Base1 (anticipate tasks)	1.1	1.1
Base2 (predict actions)	1.3	1.2
Base3	1.4	1.4
Base8 (LLM predict low-level actions)	1.5	1.5

Table 3
Precision and recall values of the anticipated tasks with and without the LLM and the validator.

Architecture	*Precision*	*Recall*
Simple statistical model	0.75	0.76
LLM without validator	0.78	0.76
LLM with validator	0.99	0.78

was provided for errors (e.g., action to grab an object without moving to suitable location) up to three times per trial; LLM was told why this choice was incorrect, and allowed to output another action. We measured the number of steps and time taken by the team to complete the selected 100 task routines.

In **Exp4**, we introduced another ad hoc agent with an incomplete knowledge base to the two agent team (ad hoc agent and human). The new agent's ASP program included only a subset of the objects (17/31), actions (4/7), and axioms (6/9 causal laws, 16/26 executability conditions). This corresponded to the absence of around $40 - 45\%$ knowledge; the agent had enough initial knowledge to perform some basic activities, while also withholding key knowledge to create gaps that limited the agent's ability to complete tasks. During task execution, the human agent periodically describes actions of the knowledgeable ad hoc agent to the ad hoc agent with missing knowledge. This agent used the procedure described in Section 3.4 to process these descriptions into ASP sorts, actions and axioms and added the validated information to its knowledge base. At the end of each episode, it also used decision tree induction to learn new axioms. We evaluated the agent's ability to learn missing knowledge across 10 episodes. We recorded the number of objects, actions, and axioms learned in each trial, and the precision and recall of learning these axioms compared with a complete program (ground truth).

In **Exp5**, we extended each episode from **Exp4** to include three runs. The first run in an episode had the same initial knowledge as in **Exp4**, but the subsequent two runs built on the knowledge for a potentially different sequence of tasks. This process was repeated for the 10 episodes; we recorded the objects, actions, and axioms learned after each run and episode, and computed precision and recall as the performance measures. To evaluate **H6** in **Exp6**, we ran 20 trials with and without the learned axioms, recording the number of steps and the time taken to complete the assigned task(s).

4.2. Experiment Results

Table 2 summarizes the results of **Exp1**. When the ad hoc agent reasoned with anticipated tasks and predicted actions, it provided the best performance with lowest number of action steps and least amount of time taken to complete task routines. For comparison, the average absolute values are 26.8 steps and 361 seconds for KAT. The accuracy of the human behavior prediction models learned by the ad hoc agent was 85%, i.e., it makes errors, but it supports rapid learning and revision. Also, reasoning with prior knowledge and the output of these predictive models significantly improves performance. When the agent used task anticipation but not the behavior prediction models (**Base1**), the number of steps and time taken increased; not considering the teammates' actions may lead the agent to waste time in executing redundant actions. These results emphasize the importance of the behavior prediction models, supporting hypothesis **H1**. When the ad hoc agent used the behavior prediction models but did not anticipate future tasks (**Base2**), performance worsened, with a further increase in the number of steps and the time taken to complete tasks. Planning jointly for the current and anticipated tasks saved time and effort. These results support **H2**. Also, when the ad hoc agent did not use task anticipation or the behavior prediction models (**Base3**), the performance worsened further. These results support **H1** and **H2**. Finally, using the LLM to directly compute a sequence of low-level actions (**Base8**) resulted in the worst observed performance, supporting **H4**. All results were statistically significant with $p < 0.0001$.

Table 3 shows the results from **Exp2**. The errors in the LLM's outputs were substantially reduced by the validator. The recall values do not change substantially as the validator did not introduce new tasks;

Figure 5: Average number of steps and time taken by the team to complete tasks with prompting strategies and/or external validator; performance measure values for baselines computed as a fraction of values for KAT in each trial, with average absolute values of 27.5 steps and 372.7 seconds for KAT.

Architecture	*Steps*	*Time(s)*
KAT (all prompting, with validator)	1.00	1.00
Base4 (no prompting, no validator)	1.21	1.15
Base5 (few-shot prompting, no validator)	1.17	1.18
Base6 (chain-of-thoughts, no validator)	1.16	1.16
Base7 (no prompting, with validator)	1.05	1.04

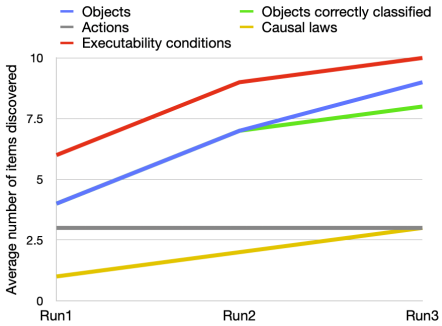

Figure 6: Average number of objects, actions, and axioms learned in three consecutive runs, averaged over 10 episodes.

Table 4
Average precision and recall of the learned axioms in 30 runs over 10 episodes.

Axiom type	*Precision*	*Recall*
Causal laws	0.96	1.0
Executability conditions	1.0	1.0

Table 5
Average steps and time taken by the team to complete task sequences expressed as a fraction baseline values.

Architecture	*Steps*	*Time(s)*
With learned axioms	0.76	0.84
Without learned axioms	1.00	1.00

it only reordered the tasks that are out of order and removed irrelevant tasks. These results support **H3**.

Table 5 shows the results from **Exp3**. We observed a significant performance improvement with the external validator and a combination of prompting methods. The results were statistically significant for **Base4-6** ($p < 0.001$); and mixed for **Base7**: reduction in steps was statistically significant ($p < 0.05$), while reduction in time was not ($p = 0.09$). This outcome matched expectations as Base7 used the validator. These results emphasize the importance of the validator and support hypothesis **H3**.

Results of **Exp4** are summarized in Table 4. We observed high precision and recall for learning the missing axioms. Figure 6 summarized the results of **Exp5**. By the end of the first run, the agent successfully learned 4-5 of 14 missing objects, all missing actions, 1-2 of three causal laws, and 6–7 of 10 executability conditions. After three runs, these increased to 8–9 objects, 2–3 causal laws, and 9–10 executability conditions. The steady increase in number of objects, actions, and axioms, along with high precision and recall, indicated the agent's ability to reliably learn new knowledge, supporting **H5**.

Table 5 summarizes the results of **Exp6**, which evaluated the impact of the learned knowledge on the team's ability to complete tasks. We ran paired trials with and without the learned axioms and computed performance measure values for the former as a fraction of the values for the latter. Reasoning with the learned knowledge substantially improved performance. Without this knowledge, at least one ad hoc agent often could not compute valid plans to complete the tasks, and could not contribute to the team's performance. The significant low number of steps and time ($p < 0.0001$) emphasize the importance of learning new knowledge, supporting **H6**. Additional results in the form of videos and execution traces are available in our open-source code repository [20].

5. Conclusions

This paper described an AHT architecture that embeds the principles of refinement, ecological rationality, and interactive learning, enabling an ad hoc agent to: leverage the generic knowledge in a pretrained LLM for high-level task anticipation; rapidly learn models predicting teammates' actions; perform non-monotonic logical reasoning with relevant prior knowledge and behavior models to plan and execute actions that jointly achieve current and anticipated tasks; and leverage natural language descriptions of action outcomes and runtime observations to acquire formal representations of previously unknown knowledge. Experiments in a realistic simulation environment demonstrated performance improvements compared with various baselines, highlighting the importance of each component and the ability to scale to additional agents. Future work will extend this approach to physical robots in AHT settings.

References

[1] P. Stone, G. Kaminka, S. Kraus, J. Rosenschein, Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination, in: AAAI Conference on Artificial Intelligence, 2010, pp. 1504–1509.

[2] R. Mirsky, I. Carlucho, A. Rahman, E. Fosong, W. Macke, M. Sridharan, P. Stone, S. Albrecht, A Survey of Ad Hoc Teamwork: Definitions, Methods, and Open Problems, in: European Conference on Multiagent Systems, 2022.

[3] X. Puig, K. Ra, M. Boben, J. Li, T. Wang, S. Fidler, A. Torralba, Virtualhome: Simulating household activities via programs, in: International Conference on Computer Vision and Pattern Recognition, 2018, pp. 8494–8502.

[4] H. Dodampegama, M. Sridharan, Knowledge-based Reasoning and Learning under Partial Observability in Ad Hoc Teamwork, Theory and Practice of Logic Programming 23 (2023) 696–714.

[5] H. Dodampegama, M. Sridharan, Reasoning and Explanation Generation in Ad hoc Collaboration between Humans and Embodied AI, in: International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR), Dallas, USA, 2024.

[6] H. Dodampegama, M. Sridharan, Reasoning with Commonsense Knowledge and Decision Heuristics for Scalable Ad hoc Human-Agent Collaboration, in: Distributed AI Conference (DAI), London, UK, 2025.

[7] M. Bowling, P. McCracken, Coordination and adaptation in impromptu teams, in: National Conference on Artificial Intelligence, 2005, p. 53–58.

[8] S. Barrett, P. Stone, S. Kraus, A. Rosenfeld, Teamwork with limited knowledge of teammates, in: AAAI Conference on Artificial Intelligence, volume 27, 2013.

[9] J. G. Ribeiro, G. Rodrigues, A. Sardinha, F. S. Melo, Teamster: Model-based reinforcement learning for ad hoc teamwork, Artificial Intelligence 324 (2023) 104013.

[10] S. Barrett, A. Rosenfeld, S. Kraus, P. Stone, Making friends on the fly: Cooperating with new teammates, Artificial Intelligence 242 (2017) 132–171.

[11] S. Chen, E. Andrejczuk, Z. Cao, J. Zhang, AATEAM: Achieving the ad hoc teamwork by employing the attention mechanism, in: AAAI, 2020.

[12] M. A. Rahman, N. Hopner, F. Christianos, S. V. Albrecht, Towards open ad hoc teamwork using graph-based policy learning, in: International Conference on Machine Learning, 2021, pp. 8776–8786.

[13] Q. Fang, J. Zeng, H. Xu, Y. Hu, Q. Yin, Learning ad hoc cooperation policies from limited priors via meta-reinforcement learning, Applied Sciences 14 (2024).

[14] L. Zintgraf, S. Devlin, K. Ciosek, S. Whiteson, K. Hofmann, Deep interactive bayesian reinforcement learning via meta-learning, in: International Conference on Autonomous Agents and Multiagent Systems, 2021.

[15] P. Xu, Y. Zhang, L. Hao, Q. Yan, DETEAMSK: A Model-Based Reinforcement Learning Approach to Intelligent Top-Level Planning and Decisions for Multi-Drone Ad Hoc Teamwork by Decoupling the Identification of Teammate and Task, Aerospace 12 (2025) 1–32.

[16] X. Liu, P. Li, W. Yang, D. Guo, H. Liu, Leveraging Large Language Model for Heterogeneous Ad Hoc Teamwork Collaboration, in: Robotics: Science and Systems, Delft, The Netherlands, 2024.

[17] M. Gelfond, D. Inclezan, Some Properties of System Descriptions of AL_d, Applied Non-Classical Logics, Special Issue on Equilibrium Logic and ASP 23 (2013).

[18] M. Balduccini, M. Gelfond, Logic Programs with Consistency-Restoring Rules, in: AAAI Spring Symposium on Logical Formalization of Commonsense Reasoning, 2003.

[19] E. Balai, M. Gelfond, Y. Zhang, Towards Answer Set Programming with Sorts, in: Conference on Logic Programming and Nonmonotonic Reasoning, 2013.

[20] H. Dodampegama, M. Sridharan, https://github.com/hharithaki/KAT-VH, 2026.

[21] M. Sridharan, M. Gelfond, S. Zhang, J. Wyatt, REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics, Journal of Artificial Intelligence Research 65 (2019) 87–180.

[22] G. Gigerenzer, What is Bounded Rationality?, in: Routledge Handbook of Bounded Rationality,

Routledge, 2020.
[23] H. A. Simon, Rational Choice and the Structure of the Environment, Psychological Review 63 (1956) 129–138.
[24] G. Gigerenzer, W. Gaissmaier, Heuristic Decision Making, Annual Review of Psychology 62 (2011).
[25] G. Gigerenzer, Towards a Rational Theory of Heuristics, Palgrave Macmillan UK, London, 2016, pp. 34–59.
[26] K. Katsikopoulos, O. Simsek, M. Buckmann, G. Gigerenzer, Classification in the Wild: The Science and Art of Transparent Decision Making, MIT Press, 2021.
[27] S. Kambhampati, K. Valmeekam, L. Guan, M. Verma, K. Stechly, S. Bhambri, L. Saldyt, A. Murthy, Position: LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks, in: International Conference on Machine Learning, 2024.
[28] G. A. Miller, Wordnet: a lexical database for english, Commun. ACM 38 (1995) 39–41.
[29] T. Mota, M. Sridharan, A. Leonardis, Integrated Commonsense Reasoning and Deep Learning for Transparent Decision Making in Robotics, Springer Nature CS 2 (2021).