Reasoning with Commonsense Knowledge and Decision Heuristics for Scalable Ad hoc Human-Agent Collaboration

Hasra Dodampegama
Mohan Sridharan
hasra.dodampegama@ed.ac.uk
m.sridharan@ed.ac.uk
Institute of Perception, Action, and Behavior, School of Informatics, University of Edinburgh
Edinburgh, UK

Abstract

AI agents in practical domains often have to cooperate with other agents without prior coordination. State of the art approaches for such *ad hoc teamwork* pose this task as a learning problem, using a large dataset to model the action choices of other agents (or agent types) and determine the actions of the ad hoc agent. These methods lack transparency and make it difficult to rapidly revise existing knowledge in response to changes. We present an architecture for ad hoc teamwork that leverages the complementary strengths of knowledge-based and data-driven methods for reasoning and learning. For any given goal, the ad hoc agent determines its actions through non-monotonic logical reasoning with: (a) prior domain-specific commonsense knowledge; (b) models learned and revised rapidly to predict the behavior of other agents; and (c) anticipated abstract future goals based on generic knowledge of similar situations in an existing foundation model. The agent also processes natural language descriptions and observations of other agents' behavior, incrementally acquiring and revising knowledge in the form of objects, actions, and axioms that govern domain dynamics. We experimentally evaluate our architecture's capabilities in *VirtualHome*, a realistic simulation environment.

Keywords

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

1 Introduction

Consider an assistive AI agent deployed to complete household tasks in collaboration with a human it has not worked with before. Figure 1 shows snapshots of a motivating scenario in which the AI agent and a human agent are preparing breakfast and setting up a workstation. The agents have a limited view of the environment and do not communicate with each other. Each agent is aware of the state of the domain, including the location of teammates and the outcomes of their actions (e.g., change in location of an object moved by a teammate). The agents have to reason with different descriptions of domain knowledge and uncertainty that include qualitative statements ("eggs are usually in the fridge") and quantitative measures of uncertainty ("I am 90% sure I saw the eggs on the table"), adapting their actions to changes in the domain and teammates' behavior. These characteristics correspond to *Ad hoc Teamwork* (AHT), that requires cooperation "on the fly" without prior coordination [26]; and arises in many practical applications in robotics, like disaster rescue and exploration.

Figure 1: Screenshots from *VirtualHome* [21] showing a human (female in green top) and an assistive AI agent (male in blue shirt) collaborating.

Research in AHT has evolved from using predefined protocols that encoded specific actions for the ad hoc agent in specific situations, to methods that learn probabilistic or deep network models to estimate the behavior of other agents (or agent "types") and optimize the ad hoc agent's actions based on a long history of prior interactions with these agents. It is often difficult to gather such datasets of different situations and computationally expensive to build the necessary models. Moreover, these models lack transparency, making it difficult to understand the agent's decisions.

In a departure from existing work, our prior work developed an AHT architecture that enabled an ad hoc agent to make decisions based on non-monotonic logical reasoning with prior knowledge and simple predictive models of other agents' behavior [6]. This paper extends our prior work to enable each ad hoc agent to:

(1) Leverage a Large Language Model (LLM) to anticipate future tasks to be completed, adapting the LLM's output to domain-specific knowledge and experience;
(2) Compute and execute plans for completing current task and preparing for upcoming tasks through non-monotonic logical reasoning with prior knowledge and rapidly-learned predictive models of other agents' behavior;
(3) Incrementally revise prior knowledge using LLM-based processing of natural language descriptions of actions and outcomes, and decision tree induction from observations.

We use Answer Set Programming (ASP) [11] for non-monotonic logical reasoning, GPT4o mini [20] as the LLM. We explored decision making at different abstractions in household scenarios in *VirtualHome*, a realistic physics-based 3D simulation environment for multiagent collaboration [21].

2 Related Work

Researchers have explored AHT for more than two decades under different names [18]. Initial work used predefined protocols or plays

that encoded specific actions for the ad hoc agent in specific scenarios [4]. Subsequent work used probabilistic and sampling-based methods such as Upper Confidence bounds for Trees to determine the ad hoc agent's actions [1, 3, 23]. Recent approaches considered state of the art for AHT have posed it primarily as a learning problem; a key component predicts behavior of other agents and determines the ad hoc agent's behavior using a long 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 [2]; attention-based recurrent neural networks for real-time adaptation [5]; graph neural networks to simulate interactions with other agents (agent types) and determine the ad hoc agent's behavior [22]; self-play to learn a cooperation policy and candidate teammate policies [9]; sequential and hierarchical variational auto encoders to model teammates' changing behaviors [28]; and model-based RL methods to learn separate models of the environment and teammates [27]. These methods are resource-hungry, requiring substantial computation and training examples to learn the models that are often opaque, limiting interpretability of the system.

Frameworks based on "foundation" models are considered state of the art for various problems in robotics and AI, and an LLM-based framework has been developed to compute the ad hoc agent's actions [16]. At the same time, it is known that such models can make arbitrary decisions in novel situations, do not plan, and are more effective when used to generate abstract guidance that is validated before use [14].

This paper builds on our proof-of-concept work [6] 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, use logics to jointly plan for current and future tasks, and use decision-tree induction and an LLM for acquiring previously unknown knowledge from observations and natural language descriptions.

3 Architecture Description

Figure 2 outlines KAT, our architecture for an ad hoc agent collaborating with other agents (human, AI). An external *task generator* is used to generate a realistic, evolving routine of tasks for any given day (e.g., "make breakfast", "set up workstation"), dispatching tasks one at a time to all agents. Each agent is unaware of the task generation strategy and starts with no prior knowledge of the preferences, capabilities, and strategies of other agents, although it expects teammates to collaborate to complete assigned tasks. Each agent receives information about the current state, and computes and executes actions to complete task(s). The ad hoc agent determines its action by reasoning with prior knowledge (Section 3.1) and the actions of other agents predicted by models learned from runtime observations (Section 3.2). It also prompts an *LLM* with recent observations and completed tasks to anticipate future tasks (Section 3.3). It validates and adapts the LLM's output based on domain-specific knowledge, and jointly plans actions to achieve the current and anticipated task(s). In addition, a human (agent) occasionally describes an agent's actions (e.g.,"Agent 1 cannot put the cake in the microwave since its door is closed"). The ad hoc agent uses these descriptions to learn previously unknown knowledge in the form of objects, actions, and axioms. It also uses observations obtained during plan execution to learn missing axioms based on decision tree induction (Section 3.4). We use the example scenario given below to describe KAT's components for one ad hoc agent and a human; we consider additional agents during evaluation.

Example 1. ***[Human-AI agent collaboration scenario]*** Consider an AI agent and a human agent collaborating to complete daily household tasks such as preparing breakfast or setting up the home work-station (see Figure 1). The agents can interact with the environment through actions that involve moving to places, picking up or placing objects, switching appliances on or off, and opening or closing appliances. Completing a task requires a sequence of such actions to be computed and executed by members of the team without any direct communication between them. The ad hoc agent assumes that any teammate will have access to the same information about domain state, predicts the actions the teammate will execute over the next few steps, and computes its plan to complete the current task and prepare for the upcoming task(s). Each ad hoc agent's prior commonsense knowledge includes relational descriptions of some attributes of the domain, objects, and human. It also includes axioms governing actions and changes, e.g., each agent is aware that it cannot hold more than two objects at a time.

3.1 Knowledge Representation and Reasoning

In KAT, any given domain's transition diagram is described using an extension of *action language* $\mathcal{AL}_d$ [10]. 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 $agent\_actions$ such as $grab(ad\_hoc\_agent, object)$, $move(ad\_hoc\_agent, loaction1, location2)$ that are performed by ad hoc agent; or $exo\_actions$ (exogenous actions) such as exo_grab $(other\_agent, object)$, and $exo\_switch\_on(other\_agent, appliance)$ which are performed by other agents, e.g., human or another AI agent. Statics (fluents) are domain attributes whose values cannot change. Fluents 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 the by ad hoc agent's actions, e.g., $agent\_at(other\_agent, location)$ is a teammate's location computed by (say) external sensors. Given a specific Σ, the domain's dynamics are described using axioms such as:

$$open(A, E) \textbf{ causes } opened(E) \tag{1a}$$

$$\neg at(A, L1) \textbf{ if } at(A, L2), L1 \neq L2 \tag{1b}$$

$$\textbf{impossible } grab(A, O) \textbf{ if } on(O, E),\ \neg opened(E) \tag{1c}$$

where Statement 1(a), a *causal law*, implies that an agent executing the $open(A, E)$ action causes an appliance E to be opened; Statement 1(b), a *state constraint*, implies that an agent (A) cannot be

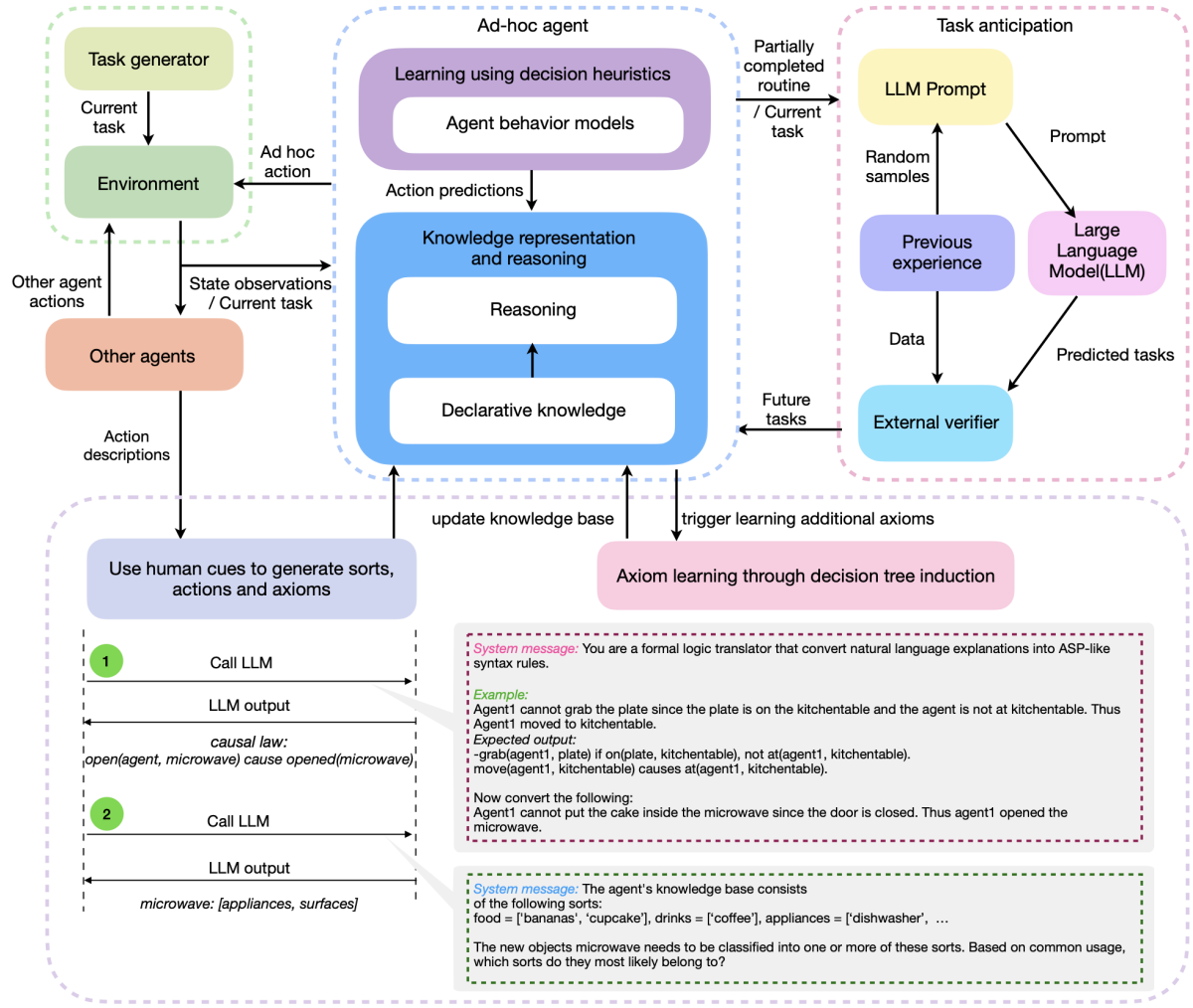

Figure 2: KAT leverage the complementary strengths of knowledge-based and data-driven reasoning and learning.

in two places ($L1, L2$) at the same time; and Statement 1(c), an *executability condition*, prevents the ad hoc agent(A) from trying to grab an object (O) from an appliance (E) that is not open. The history, $\mathcal{H}$, is a record of statements of the form $obs(fluent, boolean, step)$, which represent observations, and $hpd(action, step)$, which represent executed actions, at specific steps. $\mathcal{H}$ also includes default statements that are true in the initial state.

To reason with knowledge, a script automatically constructs program $\Pi(\mathcal{D}, \mathcal{H})$ in CR-Prolog, an extension to ASP that supports consistency restoring (CR) rules. $\Pi(\mathcal{D}, \mathcal{H})$ contains statements from $\mathcal{D}$ and $\mathcal{H}$, inertia axioms, reality check axioms, closed world assumptions for defined fluents and actions, helper relations such as $holds(fluent, step)$ and $occurs(action, step)$ that imply (respectively) that a fluent is true and an action is part of a plan at a particular time step, and helper axioms that define goals and guide planning and diagnosis. ASP is based on stable model semantics, and encodes *default negation* and *epistemic disjunction*, i.e., unlike "$\neg a$" that states *a is believed to be false*, "*not a*" only implies *a is not believed to be true*, and unlike "$p \vee \neg p$", "*p or* $\neg p$" is not tautologous. Each literal is true, false, or unknown, and the agent only believes that which it is forced to believe. ASP supports non-monotonic logical reasoning, i.e., the ability to revise previous conclusions, which is essential for agents operating in complex, practical domains. The CR rules allow the agent to make assumptions under exceptional circumstances to recover from inconsistencies. All reasoning tasks, i.e., planning, diagnostics, and inference are reduced to computing *answer sets* of Π subject to some criteria (e.g., minimize costs) and extracting the action sequence.

Our example scenario is complex, with many objects, containers, and locations, e.g., there can be $\approx 10^{25}$ states with just one ad hoc agent and one human, making it computationally expensive to compute plans comprising multiple steps. To support scalability, we build on prior work on a refinement-based architecture [25]. 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.

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 and 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)

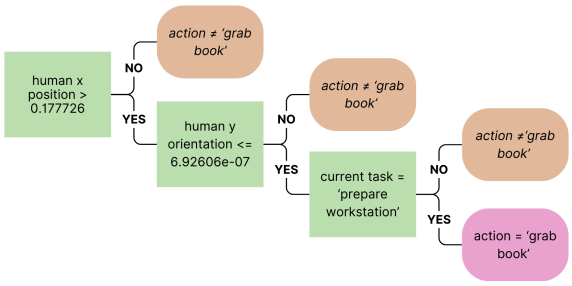

Figure 3: FF tree model of human behavior for the grab_book action in the example scenario from the *VirtualHome*.

3.2 Agent Behavior Models

Since reasoning with prior domain knowledge that is incomplete or inconsistent will lead to poor performance, KAT enables the ad hoc agent to reason with models that predict the action choices of other agents that are learned (and revised) rapidly. This capability is achieved by embedding the Ecological Rationality (ER) principle [13], which is based on Herb Simon's original definition of Bounded Rationality [24] and the algorithmic theory of decision heuristics [12]. ER explores decision making "in the wild", i.e., under open world uncertainty with the space of possibilities not fully known, and characterizes behavior as a joint function of internal cognitive processes and the environment. In the absence of comprehensive knowledge, optimal decisions may be unknowable and not just hard to compute. ER advocates the use of *adaptive satisficing* and decision heuristics (e.g., tallying, sequencing, fast and frugal methods) to ignore part of the information and make decisions more quickly, frugally, and accurately than methods with many more free parameters. This has been shown to provide better performance than more sophisticated methods in practical applications [12].

KAT enables the ad hoc agent to select relevant attributes and rapidly learn (and revise) an ensemble of *Fast and Frugal* (FF) trees from limited data 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 [15]. Each level of the tree contains an exit, allowing the agent to make quick decisions. Also, these models enable the ad hoc agents to consider the more informative attributes and stop as soon as a rational option is found. Figure 3 shows an FF tree learned for a human, and Table 1 shows the attributes used. The initial version of these trees were constructed from only 1000 traces of other agents' actions (guided by simple hand-crafted policies) and domain states. Consistent agreement (or disagreement) between observed outcomes and the predictions provided by these behavior models triggers the use of a particular model for subsequent steps, or leads to the revision of existing model(s), allowing the ad hoc agent to quickly adapt to changes in the domain or an agent's behavior.

3.3 Task Anticipation

As stated earlier, there is increasing evidence that LLMs make arbitrary decisions in novel situations. They are more effective when used as translators between natural and domain-specific languages, and to generate high-level (generic) guidance that is validated externally before being implemented by planning subroutines [14]. Building on these findings, KAT enables 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. In the absence of the LLM, the agents are informed about the target tasks one at a time. We experimentally demonstrate later that jointly planning to complete the current task and anticipated task, e.g., fetch some ingredients from the fridge for making lunch while fetching eggs for cooking breakfast, improves team performance. The agent uses a combination of prompting strategies to interact 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 partially completed task routine by selecting potential future tasks from the available list of tasks in the example scenario within the *VirtualHome* domain. 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 contextual features extracted from existing knowledge and recent observations to eliminate tasks that are invalid or irrelevant, and reorder tasks according to the human preferences. For example, the ad hoc 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, KAT enables the ad hoc agent to consider one anticipated task and the current task as the joint goal for which a plan is to be 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 in the form of objects, actions, and axioms 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 an 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 (see Figure 2) with two examples of input and expected output, the structure of the axioms (e.g., "a causes f if p"), and the query based on the cue (Line 1, Algorithm 1). The LLM maps this cue into candidate axioms using their known structure. From the LLM's output, the agent extracts actions (verbs), objects, and action preconditions and effects (Line 2). It then checks whether the new objects are already defined in $\Pi(\mathcal{D},\mathcal{H})$ (Line 3). Any new objects are sent to the LLM with a second prompt (Figure 2) based on a predefined message template, existing sorts in $\Pi(\mathcal{D},\mathcal{H})$, and the new objects (Lines 4-6). The LLM then categorizes each new object into one or more known basic sorts. If multiple basic sort labels are returned for an object, the lowest category in the sort hierarchy is used to add the new object to $\Pi(\mathcal{D},\mathcal{H})$ (Lines 7-8).

Next, the agent examines $\Pi(\mathcal{D},\mathcal{H})$ to check whether the action verb (e.g., *grab*) from the LLM's output is already defined (Line 10). 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 [17] 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 11-13). If the extracted action (or its equivalent) does not exist in $\Pi(\mathcal{D},\mathcal{H})$, the agent adds it with the lowest (ie., most specific) sort labels for arguments (Lines 14-17). This process ensures that we do not introduce the same action multiple times with different sorts. This procedure is also repeated for literals (Lines 18-25), 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)*'), replacing the ground sorts with variables and ensuring consistency between head and body (Line 26). The axiom is then added to $\Pi(\mathcal{D},\mathcal{H})$ if it does not exist (Lines 27-28).

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 [19].

1. The agent selects an action a_I from the newly learned set of actions A, and an initial state of the environment S_I. It simulates the execution of a_I in S_I in its current domain to collect information about the outcomes (e.g., end state, an inconsistent outcome).
2. An expected outcome's absence indicates the absence of an executability condition; any additional effects indicate missing causal law(s). If all observations match expectations for different actions and states, the current knowledge is considered to be comprehensive.
3. The agent responds to an inconsistency by simulating the execution of a_I in different states, extracting from the answer set and initial state all those fluent literals that have an object constant that is also in a_I. These collected literals form part of the *training* examples.
4. In the training examples, the ground terms in literals are replaced by variables, and the dataset is reformatted with the fluent literals as features and the presence of absence of inconsistency as the class label. Each training example then records the presence or absence of a fluent literal, and the presence or absence of an inconsistency, as a binary value.
5. Separate decision trees are constructed for causal laws and executability conditions, with the action as the root node, and nodes are split using features that have not been used before and are likely to result in the highest reduction in entropy.
6. Candidate axioms are generated by traversing the learned trees from the root to the leaves using only those nodes that agree with their class label up to a threshold level (90%) and contain at least a minimum percentage (2%) of the dataset.
7. Only candidate axioms that have sufficient support among the training examples (90%) are retained. The decision tree induction process is repeated to explore different subsets of the training data. Only axioms retained over multiple repetitions are lifted to the general form and added to $\Pi(\mathcal{D},\mathcal{H})$.

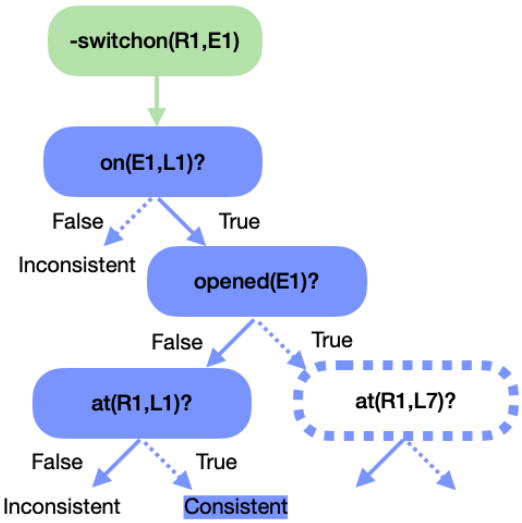

Figure 4: Part of the decision tree created to learn missing executability conditions.

Figure 4 shows part of a decision tree generated by this process, with the agent learning the following two executability conditions:

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

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

which imply that an agent cannot switch on an appliance if it is not in the same location or if the appliance's door is open. Although the strategy described above focuses on learning new objects, actions and axioms, it can be used to learn new literals (relations) too.

4 Experimental Setup and Results

We experimentally evaluated the following hypotheses regarding our architecture's capabilities:

- **H1** Reasoning with prior knowledge and rapidly-learned behavior models improves performance and promotes scalability;
- **H2** Using LLM-based anticipated tasks as joint goals improves performance compared with planning for one task at a time;
- **H3** Incrementally-updated prompts and validators improve task anticipation and team performance;
- **H4** Using LLM to directly output a sequence of 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* domain 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 our experiments, the human was modeled as a simulated entity whose action choices were based on an ASP program that considered the human's prior knowledge and runtime observations; human did not reason with models predicting teammates' actions. Also, the human's ASP program encodes certain preferences, priorities, and capabilities that are not initially known to the ad hoc agents but may be captured over time in the models they learn in order to predict the behavior of the human. The sequence of tasks generated by the task generator were assigned to the agents over time; the agents were not aware of the complete sequence and received one task at a time. The human was assigned the same goal as the ad hoc agent(s). All agents received the same observations of the domain at each step, which they used to plan their respective actions. There was no direct communication between them.

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. The agent initially assigns one learned model to each teammate but uses information from subsequent steps to incrementally revise models for each teammate based on their 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. As a result, 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.

To evaluate **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 KAT's key components. 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.

For evaluating scalability in **H1**, we increased the team size by introducing additional ad hoc agents, with three agents (one human, two ad hoc agents) and four agents (one human, three ad hoc agents) collaborating to complete tasks (in **Exp1**). These configurations would normally make collaboration increasingly challenging, e.g., with just two agents the domain has $\approx 10^{25}$ possible states, and this number increases exponentially with the number of agents. We then measured the number of steps and time taken by the agent teams to complete the tasks.

To evaluate **H3**, in **Exp2**, we computed the precision and recall of the tasks anticipated by the LLM, before and after applying the validator, over the 100 task routines. Further, we randomly selected 20 task routines and recorded the performance when the LLM was used with and without prompting methods. Baselines were: **Base4**: no prompting strategy or validator; **Base5**: few-shot prompting but no external validator; **Base6**: CoT prompting but no external validator; and **Base7**: external validator but no prompt-engineering.

For evaluating **H4**, we conducted experiment **Exp3**, in which we created an architecture that used the LLM to directly output sequences of actions for specific tasks (**Base8**). Specifically, our prompt included details of actions available in our example scenario in *VirtualHome*, their intended purpose (from ASP program, e.g., move(agent, location): move the agent to an adjacent location). We also supplied the LLM some **Action Feasibility Rules**:

- *Movement Limitation* (critical): must only move to adjacent locations defined by the next_to relationships.
- *Object Location*: must be in the same location as an object to act on it (e.g., grab, put).
- *Carrying Limit*: cannot hold more than two objects. When holding two objects, actions like open, close, switch-on, or switch-off require you to put at least one object down first.
- *Appliance Safety*: for safety, you should not open appliance doors when they are switched on.

- *Avoid Conflict*: if a human is holding an object, they will handle all actions with the object. Do not attempt to grab or interact with this object. Instead, focus on other parts of the goal.

The LLM also had access to the current world state, including the location of the agents, objects, and appliances, and each appliance's state. The problem specification also described the task to be performed; the immediate previous actions of the human and the ad hoc agent; and any specific information to be considered (e.g., human working from home). Additionally, the prompt included a detailed example of selecting an action, and asked the LLM to generate an action sequence for achieving the assigned goal.

The LLM's action choice was assigned as the ad hoc agent' action. As a recovery mechanism, we corrected errors in the LLM output up to three times per trial. For example, if the LLM's action involves grabbing an object without moving to the appropriate location, we provided feedback explaining why this choice was incorrect and allowed the LLM to predict another action for that step. We measured the number of steps and time taken by the agent team to complete the previously 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. We made sure that this 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 describe actions of the knowledgeable ad hoc agent to the ad hoc agent with missing knowledge. This agent then used the procedure described in Section 3.4 to process these descriptions into ASP sorts, actions and axioms with the help of an LLM 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 then evaluated the agent's ability to learn missing knowledge across 10 episodes, with each episode randomly selecting from five different task sequences and each sequence consisting of four tasks. Similar to previous experiments, task sequences were generated by the task generator and provided to the agent one at a time. However, we intentionally omitted the future task anticipation algorithm to prevent its influence on knowledge acquisition capabilities. 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 ASP 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 Experimental 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 tasks. The accuracy of the human behavior prediction model 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 lead the agent to waste time in executing redundant actions. These results emphasize the importance of the behavior prediction models, supporting hypothesis **H1**.

Table 2: Average number of steps and time taken to complete task routines; values for baselines computed as a fraction of these values for KAT in each trial; for comparison, average absolute values for KAT are 26.8 steps and 361 seconds.

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: Average number of steps and time taken by the teams to complete the task routines; performance measure values for Teams 2-3 computed as a fraction of these for Team 1.

Team	*Steps*	*Time(s)*
Team1	1.0	1.0
Team2	0.8	0.9
Team3	0.7	0.8

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. All results were statistically significant with $p < 0.0001$. These results support **H4**.

Next, Table 3 summarizes the performance of **Team1** (human, one ad hoc agent), **Team2** (human, two ad hoc agents) and **Team3** (human, three ad hoc agents) in completing the same set of 100 task routines. As the number of ad hoc agents increases, task completion becomes more efficient: Team2 outperformed Team1 by requiring fewer steps and less time to complete the tasks, while Team3 showed further improvements over Team2. These results emphasize the importance of efficient collaboration and demonstrate the scalability of the architecture to multiple agents, supporting **H1**. Once again all the results were statistically significant with $p < 0.0001$. The observed performance was primarily due to design choices in KAT to enable each ad hoc agent to reason independently and efficiently using domain knowledge and learned models.

Table 4: Precision and recall values of the anticipated tasks before and after applying the validator.

Architecture	*Precision*	*Recall*
Before validation (LLM output)	0.78	0.76
After validation (with validator)	0.99	0.78

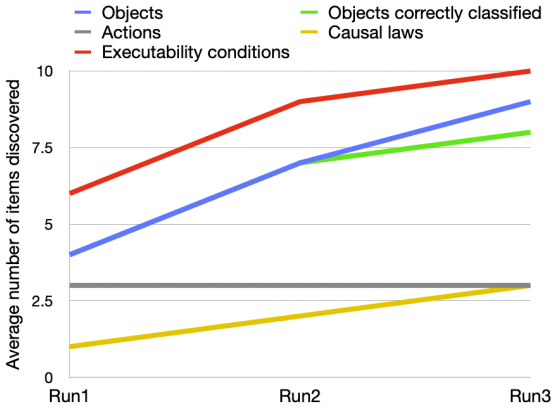

Figure 5: Average number of objects, actions, axioms learned in three consecutive runs, averaged over 10 episodes.

Table 4 and 5 shows the results from **Exp2**. We observed a marked improvement in precision after applying the validator to adapt the LLM's anticipate future tasks. The recall values do not change substantially as the validator did not introduce new tasks; it only reordered the tasks that are out of order and removed irrelevant tasks. Table 5 summarizes the performance when different prompting strategies and the validator are used with the LLM output (Section 3.3). We observed a significant improvement in performance, e.g., fewer steps and less time to complete tasks with the external validator and a combination of prompting methods. In particular, the use of the validator to adapt the LLM's output to the domain had a significant impact on performance, supporting **H3**.

Results of **Exp4** are summarized in Table 6. We observed high precision and recall for learning the missing axioms. Figure 5 summarized the results of **Exp5** with three consecutive runs in each of 10 episodes. By the end of the first run, the agent successfully learned 4-5 of 14 missing objects, all three 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 7 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. In the absence of 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 ($p < 0.0001$)

Table 5: Average number of steps and time taken by the team (human, ad hoc agent) to complete tasks with prompting strategies and/or validator; values for baselines computed as a fraction of these values for KAT in each trial, average absolute values for KAT are 27.5 steps and 372.7 seconds.

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, no validator)	1.17	1.18
Base6 (CoT, no validator)	1.16	1.16
Base7 (no prompting, with validator)	1.05	1.04

Table 6: Average precision and recall of learned axioms in 30 runs over 10 episodes; accurately learned unknown axioms.

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

Table 7: Average number of steps and time taken by the team (human, ad hoc agent) to complete task sequences represented as a fraction of these values for baseline.

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

emphasize the importance of learning previously unknown knowledge, and support **H6**. Source code, execution traces and additional results for KAT are in our **open-source repository** [8] and [7].

5 Conclusions

This paper described KAT, an architecture for Ad Hoc Teamwork (AHT) that enables an AI agent to collaborate with other agents (human, AI) without prior coordination. KAT embeds the principles of refinement, ecological rationality, and interactive learning, enabling the agent to: automatically identify and reason with information relevant to tasks at hand; leverage the generic knowledge encoded in an LLM for high-level task anticipation; rapidly learn models predicting the behavior of other agents; perform non-monotonic logical reasoning with prior knowledge and behavior models to jointly plan and execute actions to achieve the current and anticipated tasks; leverage a LLM to map natural language descriptions of action outcomes to ASP representations of previously unknown objects, actions, and axioms; and use decision tree induction to incrementally learn axioms from observations. We evaluated KAT's capabilities along with various baselines in a physics-based simulation environment, highlighting the impact of each component of KAT. Future work will explore larger teams and physical robots collaborating with humans in AHT settings.

Acknowledgments

This work was supported in part by the U.S. Office of Naval Research Award N00014-20-1-2390. All conclusions described in this paper are those of the authors alone.

References

[1] Stefano V. Albrecht and Peter Stone. 2019. Reasoning about Hypothetical Agent Behaviours and their Parameters. *CoRR* abs/1906.11064 (2019). arXiv:1906.11064 http://arxiv.org/abs/1906.11064

[2] Samuel Barrett, Avi Rosenfeld, Sarit Kraus, and Peter Stone. 2017. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence* 242 (2017), 132–171.

[3] Samuel Barrett, Peter Stone, Sarit Kraus, and Avi Rosenfeld. 2013. Teamwork with Limited Knowledge of Teammates. In *AAAI Conference on Artificial Intelligence*, Vol. 27.

[4] Michael Bowling and Peter McCracken. 2005. Coordination and Adaptation in Impromptu Teams. In *National Conference on Artificial Intelligence*. 53–58.

[5] Shuo Chen, Ewa Andrejczuk, Zhiguang Cao, and Jie Zhang. 2020. AATEAM: Achieving the Ad Hoc Teamwork by Employing the Attention Mechanism. In *AAAI*.

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

[7] Hasra Dodampegama and Mohan Sridharan. 2025. https://github.com/hharithaki/Knowledge-Acquisition.

[8] Hasra Dodampegama and Mohan Sridharan. 2025. https://github.com/hharithaki/Task-Anticipation.

[9] Qi Fang, Junjie Zeng, Haotian Xu, Yue Hu, and Quanjun Yin. 2024. Learning Ad Hoc Cooperation Policies from Limited Priors via Meta-Reinforcement Learning. *Applied Sciences* 14, 8 (2024). https://www.mdpi.com/2076-3417/14/8/3209

[10] Michael Gelfond and Daniela Inclezan. 2013. Some Properties of System Descriptions of AL_d. *Applied Non-Classical Logics, Special Issue on Equilibrium Logic and ASP* 23, 1-2 (2013).

[11] Michael Gelfond and Yulia Kahl. 2014. *Knowledge Representation, Reasoning and the Design of Intelligent Agents.* Cambridge University Press.

[12] Gerd Gigerenzer. 2016. *Towards a Rational Theory of Heuristics.* Palgrave Macmillan UK, London, 34–59.

[13] Gerd Gigerenzer. 2020. What is Bounded Rationality? In *Routledge Handbook of Bounded Rationality*. Routledge.

[14] Subbarao Kambhampati, Karthik Valmeekam, Lin Guan, Mudit Verma, Kaya Stechly, Siddhant Bhambri, Lucas Saldyt, and Anil Murthy. 2024. LLMs Can't Plan, But Can Help Planning in LLM-Modulo Frameworks. arXiv:2402.01817 [cs.AI] https://arxiv.org/abs/2402.01817

[15] Konstantinos Katsikopoulos, Ozgur Simsek, Marcus Buckmann, and Gerd Gigerenzer. 2021. *Classification in the Wild: The Science and Art of Transparent Decision Making.* MIT Press.

[16] Xinzhu Liu, Peiyan Li, Wenju Yang, Di Guo, and Huaping Liu. 2024. Leveraging Large Language Model for Heterogeneous Ad Hoc Teamwork Collaboration. arXiv:2406.12224 [cs.RO] https://arxiv.org/abs/2406.12224

[17] George A. Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (Nov. 1995), 39–41.

[18] Reuth Mirsky, Ignacio Carlucho, Arrasy Rahman, Elliot Fosong, William Macke, Mohan Sridharan, Peter Stone, and Stefano Albrecht. 2022. A Survey of Ad Hoc Teamwork: Definitions, Methods, and Open Problems. In *European Conference on Multiagent Systems.*

[19] Tiago Mota, Mohan Sridharan, and Ales Leonardis. 2021. Integrated Commonsense Reasoning and Deep Learning for Transparent Decision Making in Robotics. *Springer Nature CS* 2, 242 (2021).

[20] OpenAI et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

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

[22] Muhammad A Rahman, Niklas Hopner, Filippos Christianos, and Stefano V Albrecht. 2021. Towards Open Ad Hoc Teamwork Using Graph-based Policy Learning. In *International Conference on Machine Learning.* 8776–8786.

[23] João G. Ribeiro, Gonçalo Rodrigues, Alberto Sardinha, and Francisco S. Melo. 2023. TEAMSTER: Model-based reinforcement learning for ad hoc teamwork. *Artificial Intelligence* 324 (2023), 104013. doi:10.1016/j.artint.2023.104013

[24] Herbert A. Simon. 1956. Rational Choice and the Structure of the Environment. *Psychological Review* 63 (1956), 129–138.

[25] Mohan Sridharan, Michael Gelfond, Shiqi Zhang, and Jeremy Wyatt. 2019. REBA: A Refinement-Based Architecture for Knowledge Representation and Reasoning in Robotics. *Journal of Artificial Intelligence Research* 65 (May 2019), 87–180.

[26] Peter Stone, Gal Kaminka, Sarit Kraus, and Jeffrey Rosenschein. 2010. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In *AAAI Conference on Artificial Intelligence.* 1504–1509.

[27] Penghui Xu, Yu Zhang, Le Hao, and Qilin Yan. 2025. 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, 7 (2025). https://www.mdpi.com/2226-4310/12/7/635

[28] Luisa Zintgraf, Sam Devlin, Kamil Ciosek, Shimon Whiteson, and Katja Hofmann. 2021. Deep Interactive Bayesian Reinforcement Learning via Meta-Learning. In *International Conference on Autonomous Agents and Multiagent Systems.*

A Execution Traces

We provide some additional execution traces as a qualitative evaluation of **H3**.

A.1 Example1

Figure 6 shows an execution example where the ad hoc agent used the LLM to anticipate future tasks, with and without the prompting strategies and validator. The example was set on a weekday where the human was working from home and no guests were expected. The correct task routine in this context was: *Prepare breakfast, Prepare home work-station, Prepare coffee, Prepare lunch*. When the

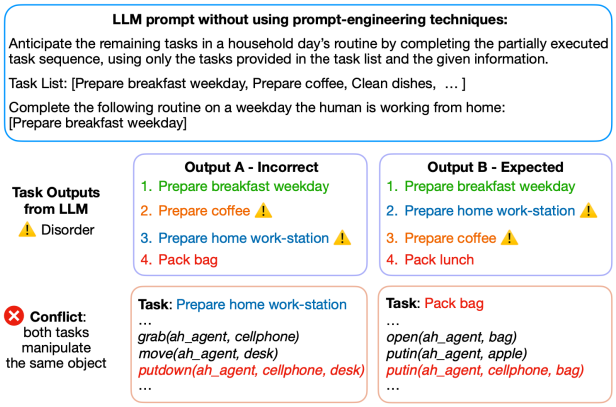

Figure 6: Execution example: using LLM without prompting strategies or external validator causes conflicts during execution, having a negative impact on performance.

ad hoc agent queried the LLM without the prompt engineering techniques or validator (Section 3.3), the anticipated task list was different from the expected output. The prompt to the LLM without using the prompt engineering strategies is shown in Figure 6. The LLM output was [*Prepare breakfast, Prepare coffee, Prepare home work-station, Pack bag*]. This output failed to align with the human preferences and priorities as:

- Higher priority was assigned to making coffee than setting up the workstation. This would delay the human for work and lead to coffee not being hot when needed.
- Packing the bag was an unnecessary task as the human was not leaving the house, and would have been filtered out by the validator.

On the other hand, when the ad hoc agent used the prompt engineering strategies and the validation strategy, the prompt to the LLM was automatically generated while incorporating context, as described in Section 3.3. The LLM's output was [*Prepare breakfast, Prepare home work-station, Prepare coffee, Prepare lunch*]. This matched the expected routine. i.e., making breakfast and setting up the workstation were considered high priority tasks, and irrelevant tasks such as *pack bag* were filtered out by the validator.

These results demonstrate the importance of using a combination of prompting techniques and the external validator, supporting **H3**.

A.2 Example2

We observed similar situations when extending the setup to three agents, one human and two ad hoc agents, collaborating on a weekend when guests were expected. The correct task routine in this context was: *Prepare breakfast, Prepare table for guests, Prepare lunch, Clean dishes.*

When the first ad hoc agent queried the LLM with the prompting strategies but without the validator, the anticipated task list by the LLM was *Prepare breakfast, Prepare table for guests, Prepare lunch, Serve snacks.* For the second ad hoc agent the LLM output was *Prepare breakfast, Prepare table for guests, Prepare lunch, Serve snacks, Clean dishes.*

When the validator was used, the outputs to both the agents were refined by incorporating context. Since the human usually did not require snacks after lunch, the task *Serve snacks* was removed. For the first agent the refined task list was *Prepare breakfast, Prepare table for guests, Prepare lunch.* For the second agent the task list was *Prepare breakfast, Prepare table for guests, Prepare lunch, Clean dishes*; since the *Clean dishes* task was a defined task in this domain, it was retained by the validator.

This example further demonstrates the importance of using the validator and support **H3**.

A.3 Example3

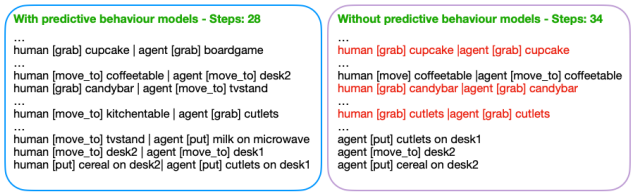

Figure 7: Execution trace for task routine: *[Prepare breakfast, Prepare activities, Serve snacks, Clean kitchen].* When an ad hoc agent is not allowed to predict and reason about the human's actions, it may choose to execute same action(s) as the human, leading to longer plans.

Figure 7 compares two plans executed by a team comprising a human and an ad hoc agent for completing a different set of tasks: *[Prepare breakfast, Prepare activities, Serve snacks, Clean kitchen]*, with and without the behavior prediction models (Section 3.2). In the first plan, when the ad hoc agent used the behavior prediction model to predict the future actions of the human, the team successfully completed all tasks for the given day in just 28 steps. On the other hand, when the ad hoc agent did not use behavior prediction models, it often selected the same actions as the human for any particular task, leading to unnecessary delays in completing the tasks. For example, in the second plan the agent frequently selected the same action as the human—simultaneously picking up the cupcake, candy bar, and cutlets, introducing redundant behavior and prolonging task execution. As a result, the overall plan was extended to 34 steps. These results demonstrate that using the behavior prediction models enables the ad hoc agent to coordinate efficiently by avoiding action conflicts with the human. This further supports **H1**.

A.4 Example4

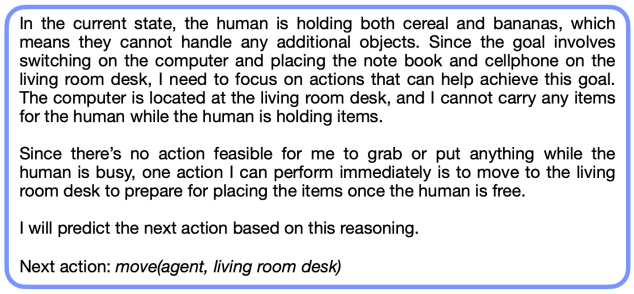

Figure 8: Example output when ad hoc agent uses the LLM for computing sequences of actions for specific tasks (Base8).

When the ad hoc agent used the LLM for directly computing sequences of actions for specific tasks **Base8**, the prompt was automatically constructed following the procedure described in Section 4.1. Specifically, it consisted of:

- a system message;
- details of the available actions in VirtualHome and their intended purposes (e.g., move(agent,location): move the agent to an adjacent location; put(agent,object,surface): place an object on a surface; open(agent,appliance): open an appliance; switchon(agent,appliance): switch on an appliance);
- action feasibility rules–see Section 4.1;
- adjacency information about locations in the domain;
- current world state (human location: living room desk, agent location: kitchen table, human holds bananas in hand, human holds cereal in hand, milk on kitchen table, apple on living room coffee table, plum on living room coffee table, mug on living room coffee table, cupcake on bedroom coffee table, candy bar on bedroom coffee table, wine on bedroom coffee table, board game on bedroom coffee table, water glass on living room desk, computer on living room desk, juice on bedroom desk, plate on counter one, cutlets on counter one, coffeemaker on counter three, coffeepot placed inside coffeemaker, book on kitchen small table, chips on kitchen small table, bread slice on kitchen table, dishwasher under counter three, cellphone on living room coffee table, dishwasher door closed, dishwasher not switched on, computer not switched on, coffeemaker not switched on);
- description on task to be performed. For example, if the team's current goal is to prepare the work-station, it has three subtasks: switch on the computer, placing the note book on the living room desk, and placing the cellphone on the living room desk;
- immediate past actions of the human and the ad hoc agent;
- contextual information about the day (weekday, and the human is working from home);
- an illustrative example of action selection;

Algorithm 1: Acquire knowledge from human cues.

Input: Human *cue*; $\Pi(\mathcal{D}, \mathcal{H})$; pretrained LLM; $systems\_msg_1$, $system\_msg_2$ (predefined); few shot prompting *examples*.
Output: Updated ASP program $\Pi(\mathcal{D}, \mathcal{H})$

$axiom \leftarrow$ LLM($system\_msg_1, examples, cue$)
$action, literals, objects \leftarrow$ parse_axiom($axiom$)
$new\_objects \leftarrow$ check_exists($objects, \Pi(\mathcal{D}, \mathcal{H})$)
if $new\_objects$ **then**
  $sorts \leftarrow$ extract_sorts($\Pi(\mathcal{D}, \mathcal{H})$)
  $sort\_categories \leftarrow$ LLM($system\_msg_2, sorts, new\_objects$)
  $obj\_categories \leftarrow$ lowest($new\_objects, sort\_categories, sorts$)
  $\Pi(\mathcal{D}, \mathcal{H}) \leftarrow$ update_core_ASP($new\_objects, obj\_categories$)
end
$act\_exists, action \leftarrow$ check_exists($action, \Pi(\mathcal{D}, \mathcal{H})$)
if act_exists with low level sorts **then**
  $action \leftarrow$ update_action_sorts($action, \Pi(\mathcal{D}, \mathcal{H})$)
  $\Pi(\mathcal{D}, \mathcal{H}) \leftarrow$ update_core_ASP($action$)
else if $\neg action\_exists$ **then**
  $action \leftarrow$ get_action_with_lowest_sorts($action, \Pi(\mathcal{D}, \mathcal{H})$)
  $\Pi(\mathcal{D}, \mathcal{H}) \leftarrow$ update_core_ASP($action$)
end
$lit\_exists, existing\_literals \leftarrow$ check_exists($literals, \Pi(\mathcal{D}, \mathcal{H})$)
for $literal \in literals$ **do**
  if $literal \notin existing\_literals$ **then**
    $literal \leftarrow$ revise_literal($literal, \Pi(\mathcal{D}, \mathcal{H})$)
    $\Pi(\mathcal{D}, \mathcal{H}) \leftarrow$ update_core_ASP($literal$)
    $existing\_literals = existing\_literals + literal$
  end
end
$axiom \leftarrow$ create_axiom($axiom, action, existing\_literals$)
if $axiom \notin \Pi(\mathcal{D}, \mathcal{H})$ **then**
  $\Pi(\mathcal{D}, \mathcal{H}) \leftarrow$ update_core_ASP($axiom$)
end

- a query asking the LLM to predict the next action required to achieve the task.

Figure 8 shows the resulting output from the LLM. The selected action move(agent, living room desk) violated the 'Movement Limitation (Critical)' rule in 'Action Feasibility Rules', which required the agent to verify the adjacency of locations before attempting to move. This example demonstrates that the LLM may not respect constraints even when they are provided as input, and highlights that an LLM is not designed for computing plans for non-trivial tasks; using an LLM to directly output a sequence of low-level actions to complete tasks can lead to poor performance, which supports hypothesis **H4**.

B Algorithm for Learning from Human Cues

We provide the algorithm referenced in Section 3.4, which the ad hoc agent used to learn new objects, actions, and axioms from cues provided by a human during task execution.